NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

FEASIBILITY OF TACTICAL AIR DELIVERY RESUPPLY USING GLIDERS

by

Chaz R. Henderson

December 2016

Thesis Advisor:	Oleg A. Yakimenko
Second Reader:	Fotis A Papoulias

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704–0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2016	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE FEASIBILITY OF TACTICAL AIR DELIVERY RESUPPLY USING GLIDERS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Chaz R. Henderson	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Due to the high cost and logistical burden placed on deployed units by the Joint Precision Aerial Delivery System (JPADS), the USMC has requested proposals for a single-use tactical resupply glider that can resupply squads with 500 pounds of essential food and gear while costing less than $3,000 per unit and using commercial-off-the-shelf (COTS) electronics. The feasibility of this request was determined by designing, constructing, and testing two prototype gliding aerial delivery systems, "Pun-Jet" and "Sparrow," using modern design and manufacturing techniques including AutoCAD, 3D printing, laser cutting and CorelDraw, and conducting field testing and subsequent analysis using MATLAB. It was determined that a low-cost, glider-based precision aerial delivery system utilizing COTS electronic components is likely a viable alternative to the parachute-based systems currently in use and can be constructed using modern manufacturing techniques. Further research should include an enlarged design, logistics, improvement of landing algorithms and navigation in GPS-degraded environments.

14. SUBJECT TERMS TACAD, Tactical Air Delivery Supply Glider, aerial delivery, autonomous aerial delivery, precision aerial delivery, Joint Precision Aerial Delivery System, JPADS, AADS, ADS, PADS, commercial-off-the-shelf, COTS, modern manufacturing, glider	**15. NUMBER OF PAGES** 135
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

FEASIBILITY OF TACTICAL AIR DELIVERY RESUPPLY USING GLIDERS

Chaz R. Henderson
Lieutenant, United States Navy
B.S., Virginia Polytechnic Institute and State University, 2010

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2016

Approved by: Oleg A. Yakimenko
Thesis Advisor

Fotis A. Papoulias
Second Reader

Ronald E. Giachetti
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Due to the high cost and logistical burden placed on deployed units by the Joint Precision Aerial Delivery System (JPADS), the USMC has requested proposals for a single-use tactical resupply glider that can resupply squads with 500 pounds of essential food and gear while costing less than $3,000 per unit and using commercial-off-the-shelf (COTS) electronics. The feasibility of this request was determined by designing, constructing, and testing two prototype gliding aerial delivery systems, “Pun-Jet” and “Sparrow,” using modern design and manufacturing techniques including AutoCAD, 3D printing, laser cutting and CorelDraw, and conducting field testing and subsequent analysis using MATLAB. It was determined that a low-cost, glider-based precision aerial delivery system utilizing COTS electronic components is likely a viable alternative to the parachute-based systems currently in use and can be constructed using modern manufacturing techniques. Further research should include an enlarged design, logistics, improvement of landing algorithms and navigation in GPS-degraded environments.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AAA	anti-aircraft artillery
AACUS	Autonomous Aerial Cargo/Utility System
AADS	autonomous aerial delivery system
ADS	aerial delivery system
AGL	above ground level
AGU	autonomous guidance unit
ALOC	air lines of communication
CIRPAS	Center for Interdisciplinary Remotely-Piloted Aircraft Studies
CONOPS	concept of operations
COTS	commercial-off-the-shelf
FBWA	Fly-By-Wire A
FOB	forward operating base
GLOC	ground lines of communication
GCS	ground control station
GPS	global positioning system
INP	innovative naval prototype
IED	improvised explosive device
JPADS	Joint Precision Aerial Delivery System
MANPADS	man portable air defense system
MEDEVAC	medical evacuation
MOE	measure of effectiveness
MOP	measure of performance
MSL	mean sea level
NPS	Naval Postgraduate School
ONR	Office of Naval Research
PADS	precision aerial delivery system
RC	radio controlled
RPG	rocket propelled grenade
RTL	Return to Launch
SAM	surface to air missile

SME	subject matter expert
UAV	unmanned aerial vehicle
USAF	United States Air Force
USB	universal serial bus
USMC	United States Marine Corps

EXECUTIVE SUMMARY

Currently, the United States military is engaged in operations that cover expansive, non-contiguous territories and are subject to an asymmetric threat. Because of this relative dispersion of forces, ground lines of communication (GLOC) can become extended requiring resupply convoys to travel long distances over enemy territory and leaving them vulnerable to attack from insurgent forces utilizing improvised explosive devices (IED), Surface-to-Air Missiles (SAM), Anti-Aircraft Artillery (AAA) and rocket propelled grenades (RPG). These threats mean that resupply missions must be conducted through air lines of communication (ALOC) using manned aircraft over hostile territory. Fielded air-dropped resupply systems have many shortfalls though: "United States Air Force (USAF) aircraft cannot meet USAF and Army accuracy standards once drop altitudes exceed 2000 feet above ground level (AGL)" (Tavan 2006, 2). The low altitude at which these drops must be conducted and the proliferation of Man Portable Air Defense Systems (MANPADS), small arms and AAA places these forces at an increased risk of attack (Tavan 2006).

Several recent advances in technology, namely the increasingly small and powerful computer systems as well as commercial-off-the-shelf (COTS) radio-controlled (RC) aircraft electronic systems marketed to the hobbyist aviation community, have presented an opportunity to increase the accuracy and decrease the cost of these systems. Also, because the United States Marine Corps (USMC) force structure is based on small, self-sustaining units operating remotely for long periods of time, these advances have generated interest in deploying units with fewer supplies and resupplying using multiple, small payloads with an air-dropped resupply system. Currently, no such system is fielded by the United States military. While the Joint Precision Aerial Delivery System (JPADS) can accomplish this task, it has a limited standoff range jeopardizing the safety of small units by broadcasting their location to the enemy and a high cost per unit that necessitates re-use in order to be a fiscally responsible option. In order to address these issues, the Marine Corps Warfighting Laboratory (MCWL) is investigating an alternative resupply system called the Tactical Air Delivery (TACAD) Supply Glider. The intent of the

program is to use commercial-grade components to develop a disposable, single-use resupply system that can provide one day of sustainment without burdening a USMC squad or disclosing their position while lowering the cost per unit by an order of magnitude (MCWL 2013, 1). The key performance parameters (KPPs) required by the MCWL are shown in Table 1:

Table 1. Key Performance Parameters of the Proposed TACAD Supply Glider. Source: MCWL (2016).

Parameter	Threshold	Objective
Air-deployable	10,000ft - 15,000ft	10,000ft – 25,000ft
L:D ratio	5:1	15:1
Cargo Impact Velocity (ft/sec)	25 – 35 ft/sec	15 - 25 ft/sec
Delivery Accuracy (CEP)	150 ft	50 ft
Payload capacity	500 lbs	700 lbs
Payload usable volume	20 cubic ft	25 cubic ft
Deployable from either [internal/external transport and release	CH-53, C-130	MV-22, CH-53, C-130
Cost	$3,000 per unit	$1,500 per unit

The objective of this thesis is to determine the feasibility of a low-cost, micro-light weight class PADS by (1) using rapid prototyping and fast and cost-effective modern manufacturing techniques to build gliding prototype airframes, (2) verifying the viability of COTS electronic components with rapid developmental and operational testing and evaluation (D/OT&E) techniques and PID gain tuning, and (3) estimating the cost of fielded systems. Two gliding ADS prototypes, a flying wing design called “Pun-Jet” and a V-tail design called “Sparrow” were developed during the design and experimentation phases of this project. While different, they were comprised of the same five components: an airframe, a release mechanism, an automated guidance system, a delivery platform, and a Ground Control Station (GCS). The PIXHAWK autopilot and GPS unit from 3DR, along with additional electronics commonplace in RC airplanes, were used in the construction of the automated guidance system. Two delivery platforms, the Arcturus T-20 UAV and the DJI Inspire quad-copter, were used to deliver the prototypes to altitude where they were released.

Based on analysis and assessment of the flight test results, the author's overall conclusion is that a low-cost, glider-based PAD system utilizing COTS electronic components as described by the MCWL is likely a viable alternative to the parachute-based systems currently in use. In addition, the COTS components are likely to provide sufficient accuracy, reliability and durability to close the capability gap and meet the operational need for rapid-response, tactical logistical resupply in austere and dispersed locations. Finally, modern manufacturing techniques are robust enough to create low-cost, fully functional gliding airframes that are both durable and reliable.

List of References

Flite Test. "FT Pun Jet Build." Accessed August 15, 2015. http://www.flitetest.com/articles/ft-pun-jet-build/

Flite Test. "FT Sparrow Build." Accessed March 1, 2016. http://www.flitetest.com/articles/ft-sparrow-build/

Marine Corps Warfighting Laboratory (MCWL). 2013. "Performance Work Statement (PWS) for Prototype Tactical Air Delivery (TACAD) Design Characterization." Accessed on October 28, 2016.

Tavan, Steve. 2006. "Status and Context of High Altitude Precision Aerial Delivery Systems." Paper presented to the AIAA Guidance, Navigation, and Control Conference and Exhibit, Keystone, CO, August 21–24.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First, I would like to thank my wife, Kathy, my parents, Kevin and Sherri, my sister, Bailey, and my “fur son,” Mordy, for their unwavering love and support; without them, none of my endeavors would be possible.

Second, I would like to thank LT Ryan Beall, USN, for his advice and assistance with this project. You are an expert in all things airborne.

Last, I would like to thank Dr. Oleg Yakimenko for taking on the role of thesis advisor and sharing his knowledge and extensive expertise during this project.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Aerial delivery systems (ADS) have been utilized by the United States Army, Air Force and Marines since the World War II era. In recent years, ADSs have been used in conflicts in Sarajevo, Bosnia-Herzegovina, Kosovo, and Afghanistan and the United States Marine Corps (USMC) has stated that the "expanded use of unmanned systems for resupply of forward-based units is not only viable, it is a critical operational requirement" (United States Marine Corps [USMC] 2013, 28).

Currently, the United States military is engaged in operations that cover expansive, non-contiguous territories and are subject to an asymmetric threat. As a result of this relative dispersion of forces, ground lines of communication (GLOC) can become extended requiring resupply convoys to travel long distances over enemy territory and leaving them vulnerable to attack from insurgent forces utilizing improvised explosive devices (IED), Surface-to-Air Missiles (SAM), Anti-Aircraft Artillery (AAA) and rocket propelled grenades (RPG). These threats mean that resupply missions must be conducted through air lines of communication (ALOC) using manned aircraft over hostile territory. However, fielded air-dropped resupply systems have many shortfalls: "United States Air Force (USAF) aircraft cannot meet USAF and Army accuracy standards once drop altitudes exceed 2000 feet above ground level (AGL)" (Tavan 2006, 2). The low altitude at which these drops must be conducted and the proliferation of Man Portable Air Defense Systems (MANPADS), small arms and AAA places these forces at an increased risk of attack (Tavan 2006).

Several recent advances in technology, namely the increasingly small and powerful computer systems as well as commercial-off-the-shelf (COTS) radio-controlled (RC) aircraft electronic systems marketed to the hobbyist aviation community, have presented an opportunity to increase the accuracy and decrease the cost of these systems. Also, because the USMC force structure is based on small, self-sustaining units operating remotely for long periods of time, these advances have generated interest in deploying

units with fewer supplies and resupplying using multiple, small payloads with an air-dropped resupply system.

No such system is currently fielded by the United States military. The capability gap that exists, the ability to minimize the risk to personnel, equipment and mission when conducting logistics support operations is addressed by the Office of Naval Research (ONR) in their Autonomous Aerial Cargo/Utility System (AACUS) Innovative Naval Prototype (INP) Concept of Operations (CONOPS). The CONOPS states the following requirements:

> The general air vehicle type is expected to operate at high density altitudes (greater than 12,000 ft density altitude), delivering multiple in-stride cargo drops over round trip distances with a threshold of 150 nautical miles and an objective of 365 nautical miles, therefore reducing the number of ground transport delivered items.
>
> The air vehicle should be one that can carry a threshold of 1600 lbs and an objective of 5000 lbs of payload internally (with some internal capacity for casualty evacuations).
>
> The air vehicle is required to travel at speeds of 110 knots threshold and 250 knots objective. Within the terminal area of 5 nautical miles, the air vehicle should be able to descend and land within a threshold of 4 minutes and an objective of 2 minutes and execute an autonomous landing as close to the requested site as possible (<1 m error from computer-designated landing site center point) without over-flight of the landing zone (i.e., the vehicle executes a straight-in approach).
>
> In addition, the air vehicle shall be able to operate at night (24/7), over harsh terrain, and in all types of environments (weather conditions to exceed manned flight capabilities, satellite-denied). (Office of Naval Research [ONR] 2012, 4–5).

B. OBJECTIVES

The objective of this thesis is to determine the feasibility of a low-cost, micro-light weight class PADS by (1) using rapid prototyping and fast and cost-effective modern manufacturing techniques to build gliding prototype airframes, (2) verifying the viability of commercial-off-the-shelf (COTS) electronic components with rapid developmental and operational testing and evaluation (D/OT&E) techniques and PID gain tuning, and (3) estimating the cost of fielded systems. The prototypes should

provide sufficient accuracy and reliability to support rapid, tactical logistics resupply to troops supporting missions in dispersed locations and subject to hostile environments by building and testing small prototype unmanned aerial vehicles (UAV).

C. CURRENT STATE

The Snowflake ADS, which was started in 2008, is an ongoing research project at the Naval Postgraduate School (NPS) that prototypes potential airdrop resupply systems for testing and analysis. All of the prototype systems utilize COTS sensors including global positioning system (GPS) units, accelerometers, magnetometers, servos and commercially available software to deliver small payloads under guided and controlled descent to a designated landing position. Until now, all of the systems under the Snowflake program have utilized para-foils to facilitate payloads of different sizes. Since there is the need for increased stand-off ranges required by the ONR CONOPS detailed in Section A of this chapter, gliders are now being considered for the same mission and are, therefore, a part of the Snowflake program.

1. Capability Gap

The USMC has requested that research into the use of unmanned systems to deliver supplies to the front lines while minimizing or eliminating the risk to human life (USMC 2013, 28). The currently fielded systems, specifically the Joint Precision Aerial Delivery System (JPADS), are exceedingly large, heavy, and expensive. The USMC has determined that these do not fulfill the urgent needs of the warfighter. Also, the ONR CONOPS addresses the following two capability shortfalls:

> Executing resupply is significantly challenging due to primarily the lack of paved roads coupled with difficult, mountainous terrain which has diminished the effectiveness of traditional means of overland logistics movement using ground transportation. The Joint Force needs an alternate means to provide sustained, time sensitive, logistics support over widely dispersed locations. Combat in urban environments has shown that moving a casualty can be difficult and time consuming. Moving an individual only a few hundred yards can take an hour or more. The extended lines of communication between forces and their forward operating bases (FOBs) (inclusive of Medical Evacuation (MEDEVAC)

by aircraft) are at risk of enemy ambush or improvised explosive device (IED) attack (ONR 2012, 3).

2. Constraints

The principal constraints of the Gliding ADS Prototype system are that it must reduce the threat presented by hostile forces to friendly assets and personnel by increasing the stand-off range of the delivery platform and reducing the cost of the system per use. While the ONR CONOPS advocates a solution that can be autonomously landed and re-launched, a single-use ADS would better suit the USMC force structure by reducing the weight carried by each Marine and enabling for faster entry and egress to and from the battlefield.

D. DESIRED STATE

The desired capability resulting from this research is a prototype gliding ADS that can, if enlarged, deliver small payloads autonomously to a predetermined landing point using COTS electronic systems and software and minimizing, to the maximum extent, cost per unit. While it is not necessary for a fielded Gliding ADS Prototype system to be reusable, it is advantageous to use the same prototype airframe for multiple tests of the system.

E. RESEARCH METHODOLOGY

Research was conducted studying open-source government documentation, having discussions with PADS subject matter experts (SME), laboratory experimentation in Watkins Hall on the campus of the Naval Postgraduate School (NPS), Monterey, CA, and flight experimentation at McMillan Airfield, Camp Roberts, CA.

F. THESIS ORGANIZATION

To address the objectives and research questions detailed in Section I, this thesis is arranged as follows:

- Chapter II includes the fundamentals of aerial delivery systems, descriptions of the Joint Precision Air Drop System (JPADS) and Blizzard autonomous aerial delivery system (AADS), and an overview of the proposed TACAD Supply Glider and a description of measures of

effectiveness (MOEs) and measures of performance (MOPs) used to assess it.

- Chapter III details the design of the components of the gliding ADS prototype. It includes a description of the Arcturus T-20 and DJI Inspire UAVs, overview of the Snowflake ADS including the design of the autonomous guidance unit (AGU) as well as the design and construction of the airframes and the deployment mechanism.

- Chapter IV compiles the Gliding ADS Prototype simulation and test results. It includes an analysis of the failure modes encountered during flight experimentation, methodology used for conducting coordinate transformation and analysis of representative flight test results

- Chapter V provides conclusions and recommendations for further research. It includes an assessment of the incorporation of low-cost technology in the development of a micro-light weight class PADS and preliminary design as well as several technical recommendations for potential improvements to the Gliding ADS Prototype.

THIS PAGE INTENTIONALLY LEFT BLANK

II. RELATED WORK

A. PREVIOUS RESEARCH

There have been significant advances in the past 50 years in the field of precision aerial delivery due to the use of controlled, gliding ram-air parachutes instead of the conventional uncontrolled round parachutes. The need to deliver various payload sizes in an increasing constrained, remote battlefield environment has led to the exploration of new applications and the design and fielding of dozens of PAD systems (Yakimenko 2015, 1). The following summary of related works includes a summary of the terms used to describe aerial delivery systems, history and development of JPADS, a description of a previous NPS research endeavor Blizzard AADS, a proposed set of MOEs and MOPs used to evaluate PADS effectiveness, and a description of the proposed Tactical Air Delivery (TACAD) Supply Glider.

B. FUNDAMENTALS OF AERIAL DELIVERY SYSTEMS

PADS are often classified by their glide ratios (L/D), or the amount of lift generated by a wing or parachute divided by the aerodynamic drag it creates while moving through the air. A higher glide ratio, in non-powered vehicles, typically equates to a greater forward distance travelled by an airframe in a standard unit of vertical descent. Categories of PADS based on this classification include Low Glide with an L/D of less than 1.5, Mid-Glide with a L/D between two and 3.5, and High-Glide with L/D greater than six. Brown and Benney also give examples of each category:

> Low-glide types include controlled conventional cargo parachutes and single-surface gliding type parachutes. At this time only the former is represented. Mid-glide types are exclusively ram-air inflated double surface, rectangular planform wings, commonly known as parafoils. High-glide types include deployable high aspect ratio wings. Several examples have been demonstrated, but none are technically mature or operational at this time. (2005, 2)

Examples of these categories are shown in Figure 1.

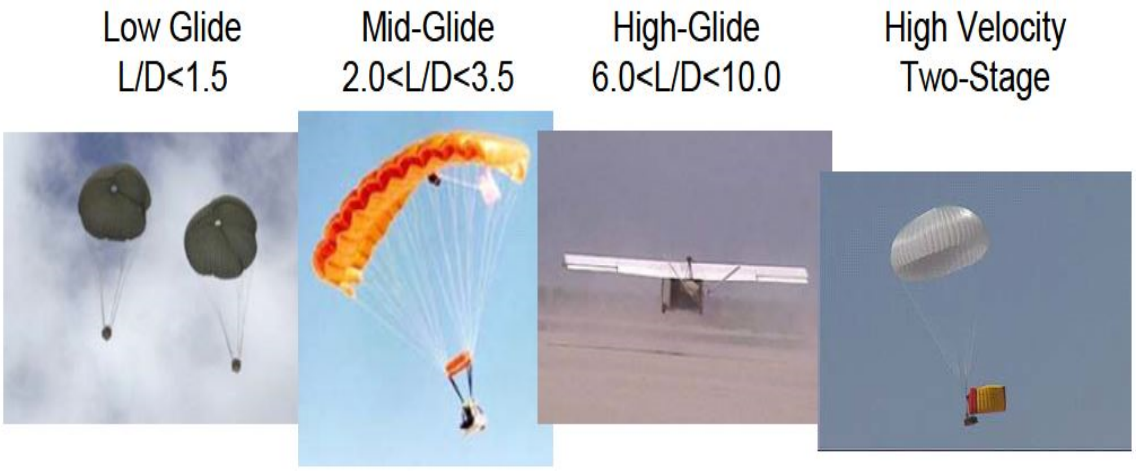

Figure 1. Categories of Precision Aerial Delivery Systems. Source: Brown and Benney (2005).

Brown and Benney also describe the following terms, also shown in Figure 2, in order to compare the performance and characteristics of different PADS:

- Impact point (IP): designated point of intended landing
- Point of impact (PI): actual point of landing
- Air release point (ARP): point of release of the airdrop unit from the drop aircraft
- Ballistic trajectory: trajectory along which an unguided, drag-only body would fall in order to reach the IP
- Ballistic ARP: the intersection of the delivery aircraft flight path with the ballistic trajectory, i.e., a theoretically perfect release point
- Calculated ARP (CARP): Standard airdrop terminology for the calculated location of the ARP based on estimated winds
- Air release circle (ARC): a circle at the release altitude, centered on the Ballistic ARP, within the glide performance of the system is sufficient to reach the IP. (2005, 4)

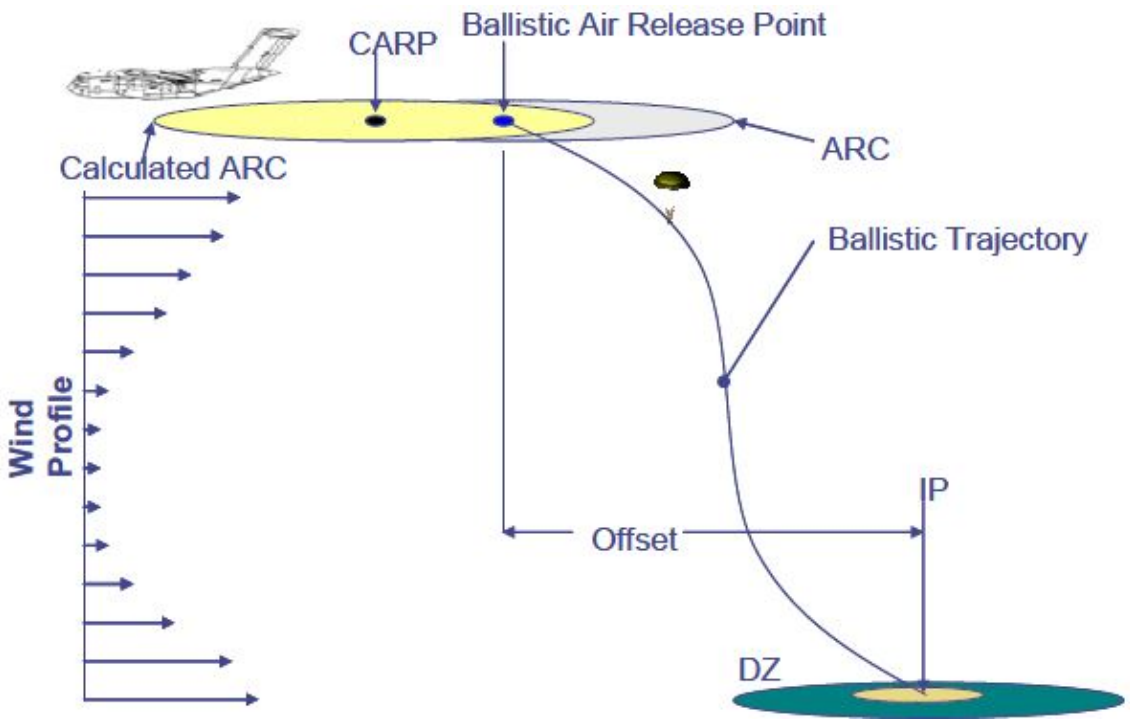

Figure 2. Terms Associated with Aerial Delivery Systems. Source: Brown and Benney (2005).

C. BLIZZARD AADS

The NPS Aerodynamic Deceleration Systems Center (ADSC) and researchers from the University of Alabama conducted experiments in 2008 to improve the performance of autonomously-guided ram-air parafoils in ultra-light-weight systems and, therefore, improve the accuracy of payload delivery of PADS. With the support of U.S. Special Operations Command (SOCOM), they developed a miniature prototype named Blizzard AADS. The Blizzard AADS consisted of four major components:

- The Arcturus T-20 UAV described in Chapter III of this paper which delivered the payload to the pre-determined drop position.
- The Snowflake ADS, shown in Figure 3, which is a 4”x8”x10” pelican case containing a GPS receiver, three-axis accelerometers, a gyroscope, a magnetometer, a barometric altimeter, and control actuators.
- A ground mission command and control center (MCCC), shown in Figure 4, for mission planning, target entry and launch/recovery operations, and

- An optional ground target weather station to facilitate target wind data generation for use in the landing algorithm (Yakimenko et al. (2011, 2–4).

Figure 3. The Snowflake ADS, part of the Blizzard AADS. Source: Yakimenko et al. (2011).

Figure 4. Blizzard AADS Mission Command and Control Center (MCCC). Source: Yakimenko et al. (2011).

While the weight of the Snowflake system is only 4.3 pounds, it is capable of carrying an additional three pounds of payload (Yakimenko et al. 2011). Also, Blizzard AADS uses the inertial trajectory in its algorithm which facilitates "accounting for the expected winds, planning and accurately tracking the final standard-approach-pattern maneuver and landing safely into the winds" (Yakimenko et al. 2011, 2). The article also states that during the first experimental drops in 2008, the Blizzard AADS achieved a circular error probable (CEP), or radius from the desired landing point in which 50% of the landings occurred, of 55 meters but, with advancements made to the guidance and control algorithms in 2009, the CEP improved to 10 meters, which is well below the 100 meter accuracy required of larger platforms and systems. While the Blizzard AADS is a ready, proven system, it best represents a research platform to prove new concepts and the NPS team has been researching the future applications shown in Figure 5 since 2011 (Yakimenko et al. 2011).

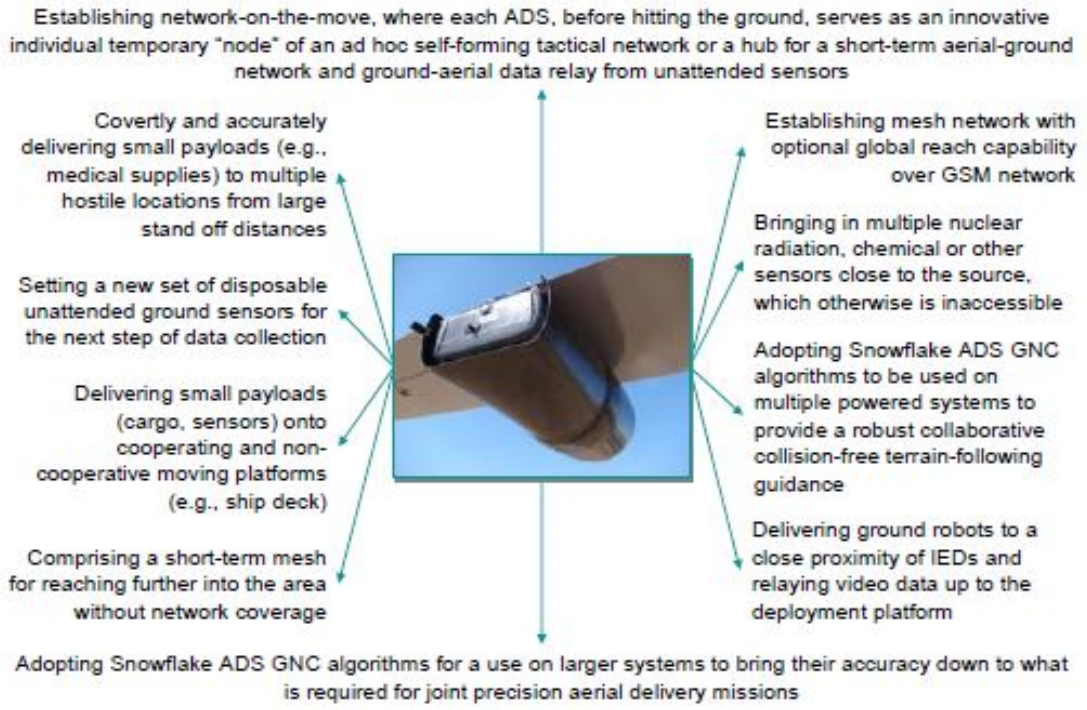

Figure 5. Possible Blizzard AADS Future Applications. Source: Yakimenko et al. (2011).

D. JPADS PROGRAM

The Joint Precision Air Drop System (JPADS), a joint program started in 1997, is a conglomeration of systems "consisting of self-guided cargo parachute systems (Army lead) and a laptop-based mission planning (MP) and weather system (USAF lead) with numerous additional partners" (Benney et al. 2007, 1). The purpose of the system is to logistically sustain combat power using high altitude, precision air drop directly into theater where the battlespace is remote, dynamic, and unsecured. An example of this system is shown in Figure 6.

The self-guided parachute system consists of a battery, GPS receiver, guidance, navigation and control software, and one of five different types of ram-air parachutes chosen depending on payload weight as described in Table 1 (*Defense Industry Daily* 2016). According to *Defense Industry Daily*, the Mission Planner system consists of a high-pressure tolerant laptop, an interface processor and dropsondes, which are hand-sized, parachute driven wind sensors.

Figure 6. JPADS in Use over Afghanistan.
Source: *Defense Industry Daily* (2016).

Mission planning takes place onboard the delivery aircraft. The software calculates or receives, from onboard sensors, the "aircraft position, altitude, airspeed, heading, ground speed, course, onboard load position (station), roll-out/exit time, decelerator opening time and trajectory to stabilization, descent rate due to weight and decelerator drag, and the descent trajectory to the desired point(s) of impact due to the atmospheric three-dimensional (3D) wind and density field encountered by the descending load under canopy" (Benney et al. 2005, 7). It then uses that information to calculate a CARP and a Launch Acceptance Region (LAR) as well as "drop and target altitudes, steering waypoints (if applicable), and weather/wind magnitudes and directions as a function of altitude, opening altitudes, and GPS 'hot start' information" (Benney et al. 2007, 7). This information can be updated in real time using dropsondes, opening altitudes and GPS information and transfers the plans to the parachute system using secure wireless or hard-wired communications. After the payload is released, the AGU

guides the system in accordance with the loaded plans and automatically corrects its heading and, to a lesser extent, altitude via actuators attached to the parachute rigging (*Defense Industry Daily* 2016).

Table 1. JPADS Parachute Categories. Source: Yakimenko (2015).

JPADS Weight Class	Weight Range
Micro-light weight (ML)	5-70 kg (10-150 lb)
Ultra-light weight	100-300 kg (250-700 lb)
Extra-light weight (XL)	300 kg-1.1 tons (700-2,400 lb)
Light weight (L)	2.3-4.5 tons (5,000-10,000 lb)
Medium weight (M)	4.5-19 tons (10,000-42,000 lb)

In addition to the resupply of troops on remote, contested battlefields, the JPADS technology has the following military and security applications:

- provides accurate and flexible stealth supply to special forces teams
- provides navigational guide for team night insertion
- supports pathfinder operation
- deploys acoustic sensing equipment into battlefield
- deploys electronic warfare equipment
- delivers leaflets accurately
- deploys nuclear, biological and chemical threat sensors
- provides "just in time" supply of advancing troops (Yakimenko 2015, 10–11)

Additionally, Yakimenko proposes the following non-military applications of the JPADS technology:

- space items recovery (as a final stage of a multistage system)
- regular supply of remote locations
- humanitarian aid and disaster relief deployment to inaccessible locations and unprepared DZs including potential field hospitals, refugee camps and United Nations compounds

- all-weather equipment drop for search and rescue operations
- equipment supply to first responders in disaster areas
- equipment delivery into rugged mountain areas
- sensing equipment and video/radio uplink deployment
- medical equipment supply
- precision delivery of buoys and lifeboats at sea (2015, 11)

Although JPADS was initially a joint effort by the U.S. Army and USAF, the Marine Corps Warfighting Laboratory (MCWL) was the first to purchase prototype PAD systems for evaluation (Yakimenko 2015). That system, the Sherpa 540kg (1,200lb) PADS, under development by Mist Mobility Integrated Systems Technology (MMIST), Incorporated, initially cost the MCWL $68,000 per unit that included the "body, canopy, riggings, remote control, rechargeable batteries and software" (Yakimenko 2015, 12). According to Yakimenko, that price rose to $100,000 per unit after 2001 when the MCWL bought 20 units to be deployed in theater. That is a drastic cost increase from the $11,000 standard military cargo parachute (Yakimenko 2015, 12) and the most recent cost point of $30 million for 110 units or nearly $275,000 per unit (*Defense Industry Daily* 2016). The high cost of these systems makes it necessary for them to be reused multiple times so they stop being cost prohibitive. Therefore, deployed units are required to disassemble, clean and repack the JPADS units for reuse. This requirement places an unnecessary burden on deployed units and prohibits JPADS' use by small units in remote, unsecured battlefields where time and manpower are limited.

E. TACTICAL AIR DELIVERY (TACAD) SUPPLY GLIDER

The USMC force structure is moving toward small detachments landing in dispersed, remote locations along the coast and then moving inland toward their objectives. Traditional lines of communication are vulnerable to attack from improvised explosive devices, surface-to-air missiles, anti-aircraft artillery and rocket propelled grenades among other weapons, so it is imperative that a new method be devised to maintain integrated logistically with the supporting elements of the Marine Air Ground

Task Force (MAGTF) and Naval Expeditionary Strike Group (ESG). While JPADS can accomplish this task, it has a limited standoff range that may jeopardize the safety of small units by broadcasting their location to the enemy and a high cost per unit that necessitates re-use in order to be a fiscally responsible option. In order to address these issues, the Marine Corps Warfighting Laboratory (MCWL) is developing a system, with the help of several contracted companies, currently called the Tactical Air Delivery (TACAD) Supply Glider. The intent of the program is to use commercial-grade components to develop a disposable, single-use resupply system that can provide one day of sustainment without burdening a USMC squad or disclosing the squad's position while lowering the cost per unit by an order of magnitude (MCWL 2013, 1).

An Operational View (OV-1) for the TACAD Supply Glider is shown in Figure 7. The concept is that it will be pre-loaded onto a CH-53 Super Stallion, C-130 Hercules or MV-22 Osprey aircraft and deployed by either internal drop out of the rear of the aircraft or by external drop via a tethered line depending on the airframe and availability of space. Multiple gliders will be loaded on each aircraft and deployed in sequence in order to reduce the flight time of the manned aircraft and, therefore, reduce the operations and maintenance (O&M) monies spent per aircraft. After deployment, wings will unfold and the glider will autonomously fly to a predetermined landing position using commercial-grade electronics and software including GPS and/or an inertial navigation system.

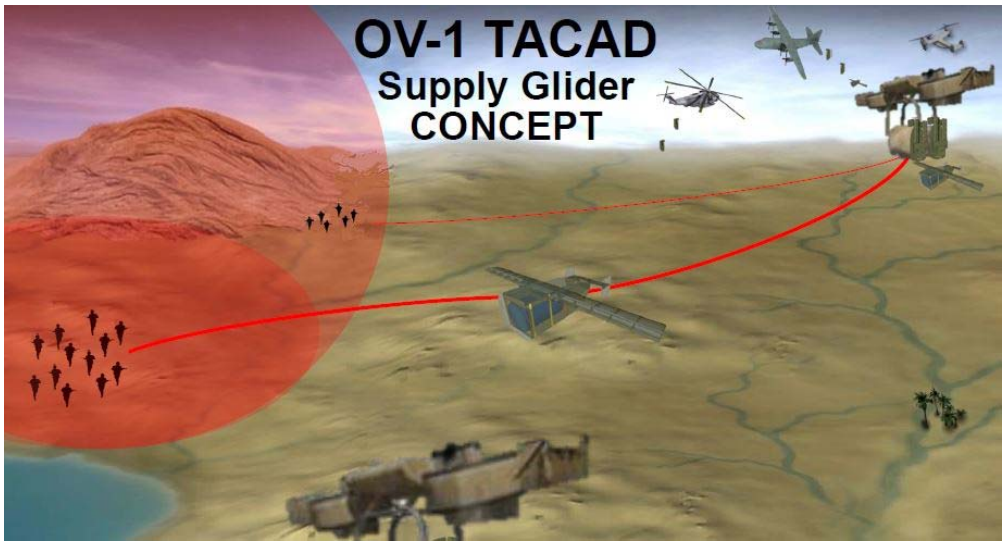

Figure 7. TACAD Supply Glider Operational View (OV-1).
Source: MCWL (2015).

Two contracted companies have submitted proposals to the MCWL and are in still in the prototyping phase. Both designs are required to meet or exceed the key performance parameters specified in Table 2. Both designs are expected to cost around $3,000 per unit that will drastically reduce the resupply costs compared to JPADS and will also make them a disposable asset and reduce the logistical burden on deployed USMC units (MCWL 2015).

Table 2. Key Performance Parameters of the Proposed TACAD Supply Glider.
Source: MCWL (2016).

Parameter	Threshold	Objective
Air-deployable	10,000ft - 15,000ft	10,000ft – 25,000ft
L:D ratio	5:1	15:1
Cargo Impact Velocity (ft/sec)	25 – 35 ft/sec	15 - 25 ft/sec
Delivery Accuracy (CEP)	150 ft	50 ft
Payload capacity	500 lbs	700 lbs
Payload usable volume	20 cubic ft	25 cubic ft
Deployable from either [internal/external transport and release	CH-53, C-130	MV-22, CH-53, C-130
Cost	$3,000 per unit	$1,500 per unit
Cost per Pound	$6.00	$2.14

THIS PAGE INTENTIONALLY LEFT BLANK

III. GLIDING ADS PROTOTYPE COMPONENTS, DESIGN AND CONSTRUCTION

Two gliding ADS prototypes, the Pun-Jet and Sparrow (Flite Test 2016), were developed during the design and experimentation phases of this project. While different, they were comprised to the same five components: an airframe, a release mechanism, an automated guidance system, a delivery system, and a Ground Control Station (GCS). Each of these components is described in detail in this chapter.

A. AIRFRAME

Two different airframes were developed in support of this project. First, a flying wing design called the Pun-Jet was obtained from the Flite Test website and modified in order for the fuselage to carry the PIXHAWK PX4 autopilot and the custom-designed release mechanism described in section B of this chapter. The line drawings of the Pun-Jet Gliding ADS are shown in Figure 8.

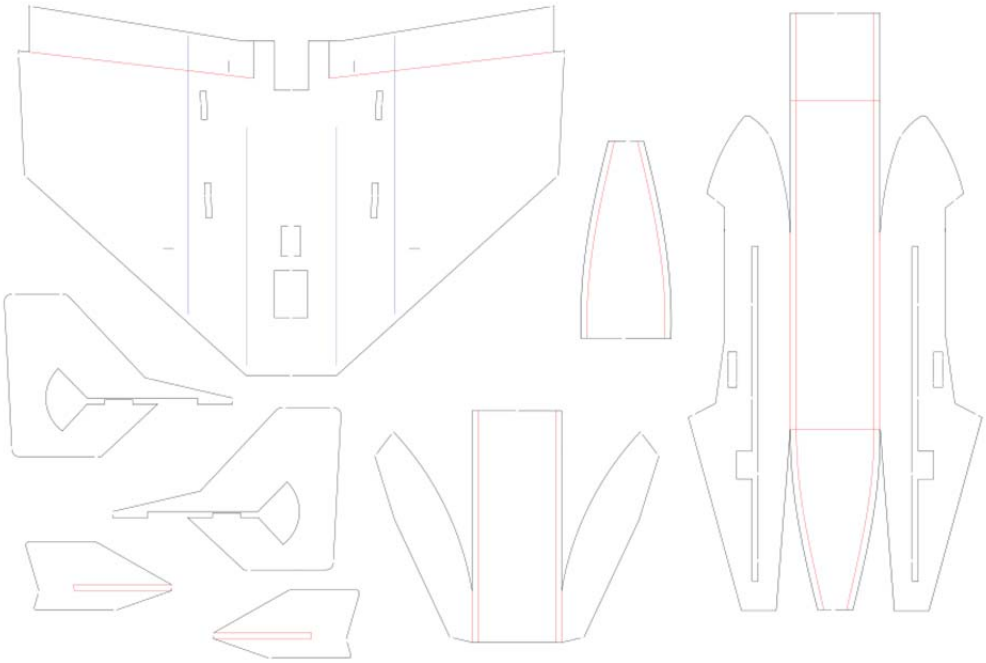

Figure 8. Pun-Jet Gliding ADS Prototype Line Drawing Plans. Adapted from Flite Test (2015).

This compact design was chosen because its delta-wing plan-form has a very low aspect ratio (Beall and Henderson 2016). In addition, stable flight is obtained using a minimal number of servos and its short wingspan ensures minimal interference with the delivery systems described in Section D of this chapter.

Secondly, a V-tail design called the Sparrow was again obtained from the Flite Test website and modified in order for the fuselage to carry the PIXHAWK PX4 autopilot and release mechanism. This airframe was chosen because of the need for an increased wing area and simplistic servo architecture. The line drawings for the Sparrow Gliding ADS are shown in Figure 9.

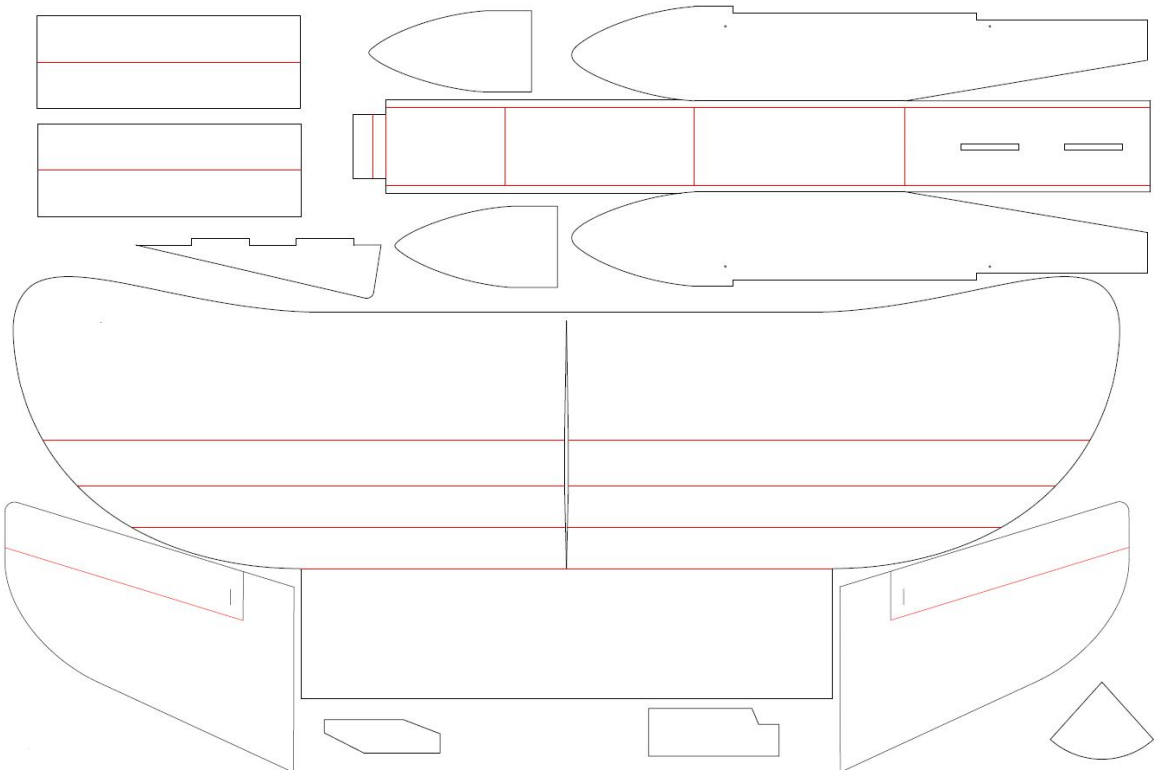

Figure 9. Sparrow Gliding ADS Prototype Line Drawing Plans
Adapted from Flite Test (2016).

An open source version of CorelDraw called Inkscape was used to modify the plans (Beall and Henderson 2016). Then, in order to ensure that the plans were scaled appropriately for the electronics and servos to be positioned as required, the plans were

printed full-size on paper. Finally, the plans were cut on a laser cutter after they were converted to a vector file. Readi-Board foam board manufactured by the R. L. Adams Plastics, Inc. company and available at Dollar Tree was used to construct both prototype airframes. The laser's power and speed can be modified relative to the color used in the plans by changing the software settings of the printer (Beall and Henderson 2016). For this project, two colors were used: black lines represented full-score cuts and red lines represented half- score cuts. Test cuts were performed on scrap foam board at various

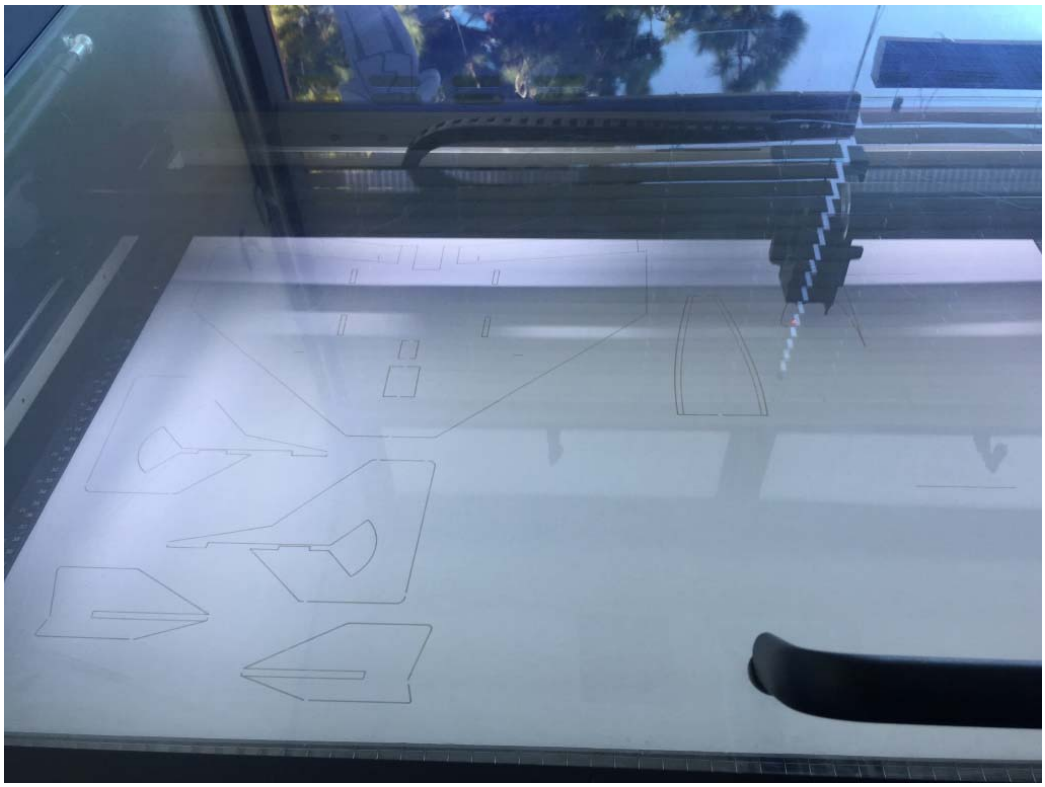

Figure 10. Pun-Jet Gliding ADS Prototype Plans Being Cut Using Spirit GLS Laser Cutter. Source: Beall and Henderson (2016).

power levels and speeds in order to determine the correct settings for the desired performance per the respective color while ensuring that the auto-focusing procedure was performed prior to cutting. For example, to ensure accurate cutting, the final power/speed settings used on the Spirit GLS laser cutter used in this project were determined to be 100% power and 50% speed for full-score cuts and 50% power / 50% speed for half-

score cuts (Beall and Henderson 2016). Figure 10 shows the plans being cut in the Spirit GLS laser cutter. When all parts are cut, the aircraft is assembled using hot glue and box tape as shown in Figure 11. Figure 12 shows the completed Sparrow Gliding ADS prototype airframe.

Figure 11. Assembling the Pun-Jet Gliding ADS Prototype Airframe.

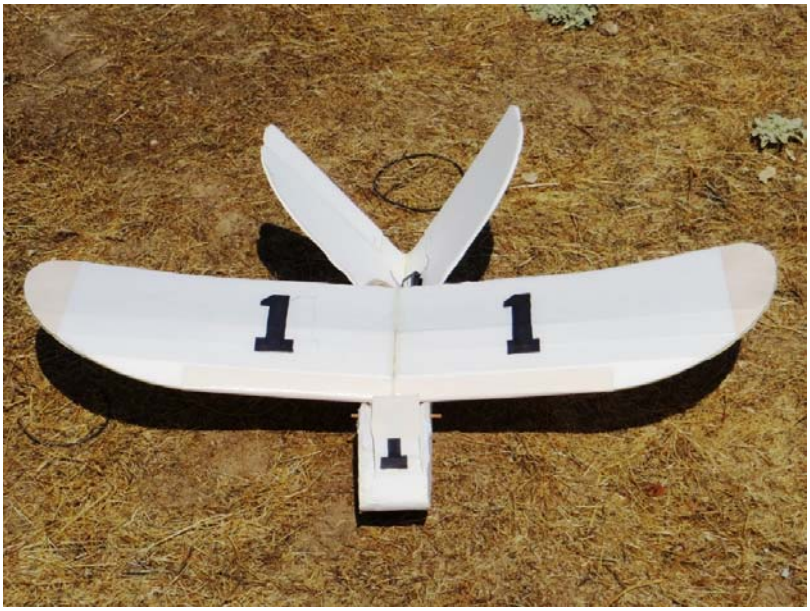

Figure 12. Completed Sparrow Gliding ADS Prototype Airframe.

B. DELIVERY PLATFORMS

Both gliding ADS prototypes were initially designed to support compatibility with the Arcturus T-20 UAV. However, it was discovered during testing that the forces generated by the launch sequence and turning flight exceeded the capabilities of the Sparrow airframe. Additionally, time between flights of the Arcturus aircraft did not support the need to gather as much data as possible. Therefore, the initial design was adapted to an additional delivery system: the DJI Inspire Quad-copter. Both systems are shown in Figure 13 and described in detail below.

Figure 13. Arcturus T-20 UAV (left) and DJI Inspire Quad-copter and Controller (right). Source: Arcturus UAV (2015).

1. Arcturus T-20 UAV

The Arcturus T-20 is a fully autonomous, single-engine fixed-wing UAV that offers extended range, altitude, endurance and payload capacity. It has been utilized throughout the Snowflake ADS and Blizzard AADS projects. The specifications of the aircraft are shown in Table 3.

Table 3. Arcturus T-20 UAV Specifications. Source: Hall (2016).

Specification	Arcturus T-20
Type	Conventional using pneumatic catapult launcher
Airframe	Airframe monocoque composite
Wing Span	17’ 6”
Length	9’ 5”
Engine	190cc 4 Stroke
Fuel	MOGAS
Typical MTOW	185 pounds
Typical Max Speed	75 knots
Endurance	10-20 hours (payload dependent)
Payload Capacity	75 pounds
Main Payload Bay	4,100 cubic inches
Rated Ceiling	15,000 feet (proven to 25,000 feet) MSL
Guidance	Fully autonomous operation, launch to landing
Characteristics	Flight and recovery under austere conditions

While the T-20 has a payload bay inside its fuselage that was used to deploy the Blizzard AADS, the wing payload mounting points, one under each side of the wing, were used to attach the Pun-Jet Gliding ADS as shown in Figure 14. The mounting points included a receptacle for a rectangular bracket that was held in place by a servo-actuated pin-in-hole mechanism.

The Arcturus T-20 was launched using a large pneumatic catapult and landed on any level surface using a skid plate mounted to the underside of the fuselage. After launch, the T-20 would take approximately five minutes to reach the deployment altitude of 2,000 feet AGL and another ten minutes to descend and land. After each flight, approximately fifteen minutes of post-flight maintenance was required before the aircraft was considered ready to fly. Therefore, approximately two experimental flights could be conducted per hour (Hall 2016).

Figure 14. Pun-Jet Gliding ADS Prototype Mounted under the Wing of the Arcturus T-20 UAV.

2. DJI Inspire Quad-Copter

The DJI Inspire is a fully autonomous, battery-powered quad-copter that is produced by the Da-Jiang Innovations Science and Technology Company based in Hong Kong. Although it was designed as an aerial photography and filmmaking platform, it was repurposed as a delivery vehicle for this project. It contains four electronic motors with carbon fiber reinforced plastic rotors that are attached to the central fuselage by carbon fiber rods. The fuselage contains the electronic controller, GPS unit, inertial measurement unit (IMU) which includes a six-axis gyroscope and accelerometer, SONAR and visual sensors, and the central processing unit (CPU). As shown in Figure 13, it is flown using a controller connected via USB to an iPad loaded with a specialized application: DJI GO (DJI 2016). The specifications of the Inspire are shown in Table 4.

Table 4. DJI Inspire Specifications. Adapted from DJI (2016).

Specification	DJI Inspire
Type	Quad-Copter
Airframe	T-600, carbon fiber and plastic
Wing Span	17.3”
Length	17.8”
Motors	DJI Model 3510H Electric
Battery	LiPo 6S 22.2V 4500mAh
Typical MTOW	7.95 pounds
Max Speed (Ascent/Descent)	5m/s, 4m/s
Endurance	18 minutes
Rated Ceiling	14,500 feet MSL (1640 feet AGL)
Guidance	Fully autonomous operation, launch to landing

This delivery system was used exclusively with the Sparrow prototype, as shown in Figure 15, but it is also compatible with the Pun-Jet.

Figure 15. Sparrow Gliding ADS Prototype Mounted on DJI Inspire

C. RELEASE MECHANISM

Previous research conducted under the Snowflake ADS and Blizzard AADS projects utilized a plastic mounting bracket, shown on the left in Figure 16, bolted to the rear of the Pelican Case and inserted into a receptacle on the underside of each wing of the Arcturus T-20 UAV. However, this design depended on the Arcturus team to deploy the system and added an additional, unnecessary weight to the gliding ADS prototypes. Therefore, an alternate design was chosen. A custom mounting bracket, shown on the right in Figure 16, containing a catch wire and a stabilization post was inserted permanently into the receptacle on the Arcturus T-20. A custom servo-activated release mechanism was designed to catch the wire using a hook in order to limit vertical motion in flight and prevent premature release of the prototype ADS and a rectangular hole to fit a stabilization post in order to prevent horizontal motion of the system in flight.

Figure 16. Snowflake ADS and Blizzard AADS Mounting Bracket (left) and Custom Gliding ADS Prototype Mounting Bracket with Catch Wire and Stabilization Post (right).

The release mechanism plate was designed to spread the launch forces across the bottom of the glider. Figure 17 shows this mechanism mounted inside the Sparrow Gliding ADS Prototype, but it was also used in the Pun-Jet prototype. The hook was attached to a dowel rod that was fed through a guiding hole on the release mounting plate. The dowel rod was then attached to a piece of a metal control arm by fishing line and super glue. The control rod was then attached to a Turnigy TGY-50090M Analog Servo. The hook would then move back and forth over the wire feed hole according to the power supplied to the servo. Figures 18 and 19 show the design of the mechanism in detail.

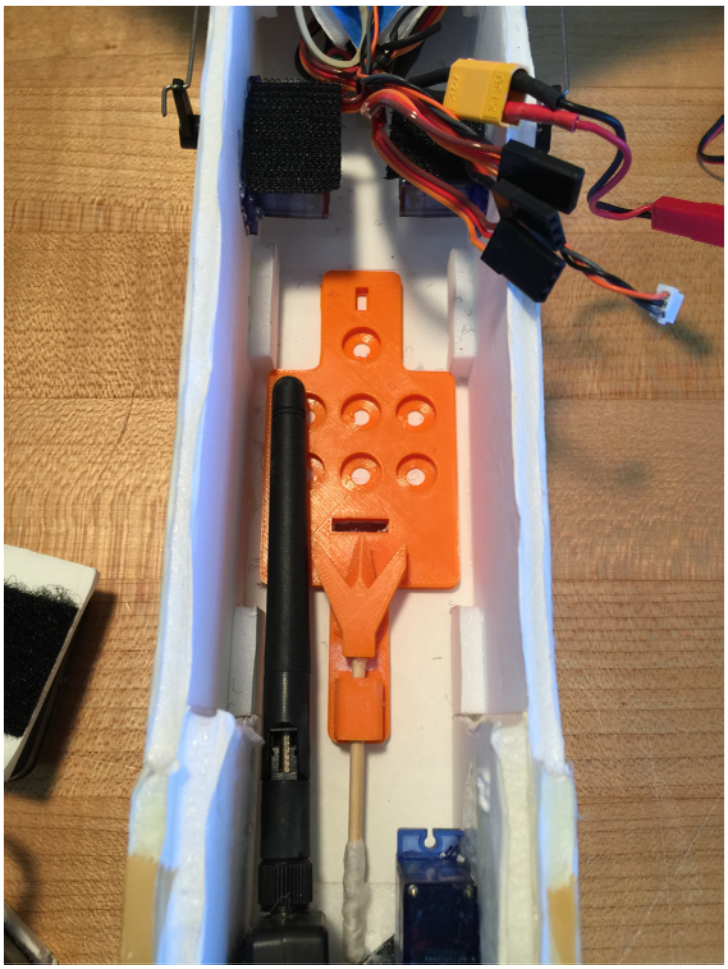

Figure 17. Release Mechanism Mounted inside Sparrow Gliding ADS Prototype.

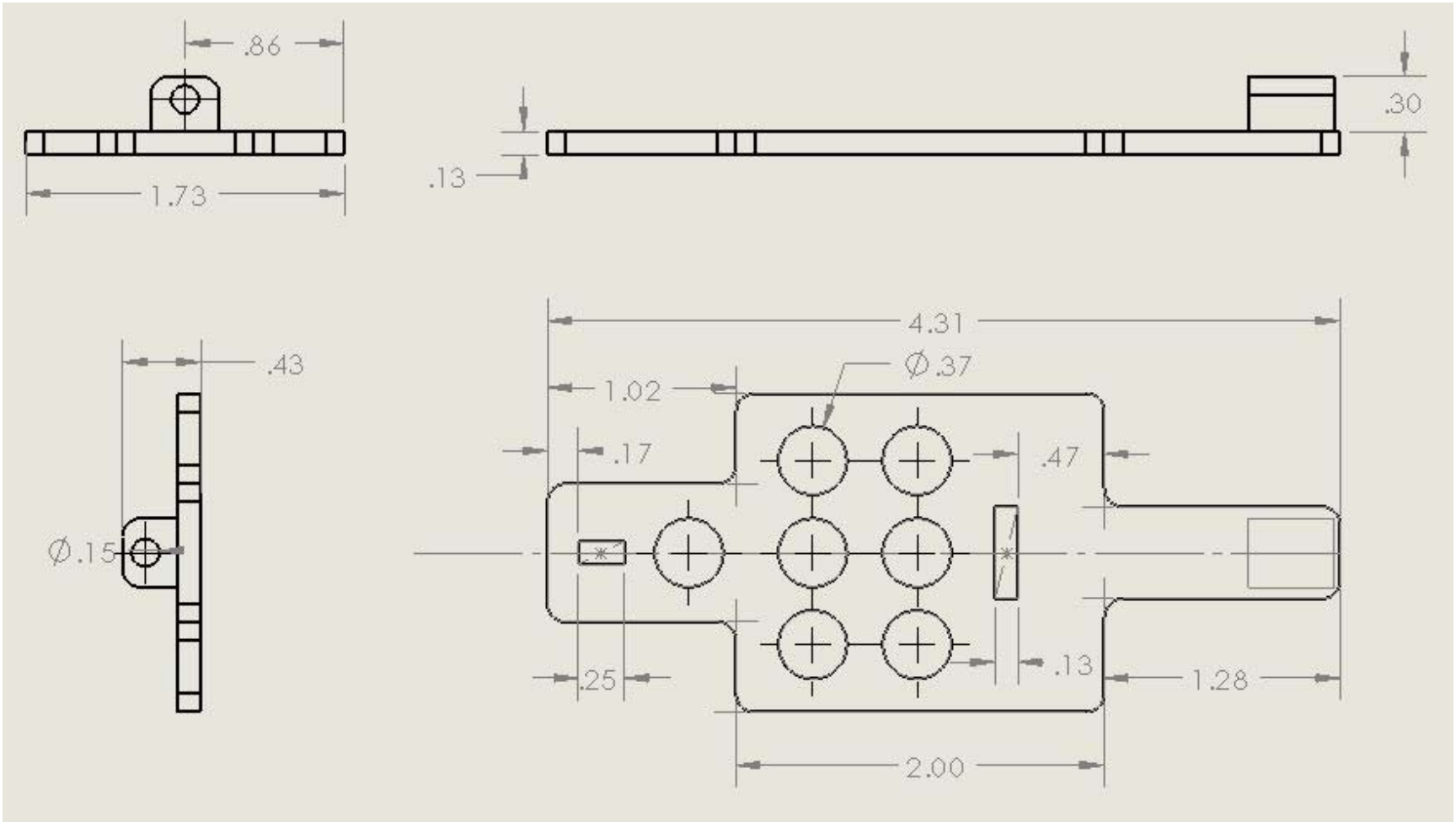

Figure 18. Detailed Drawing of the Release Mechanism Mounting Plate.

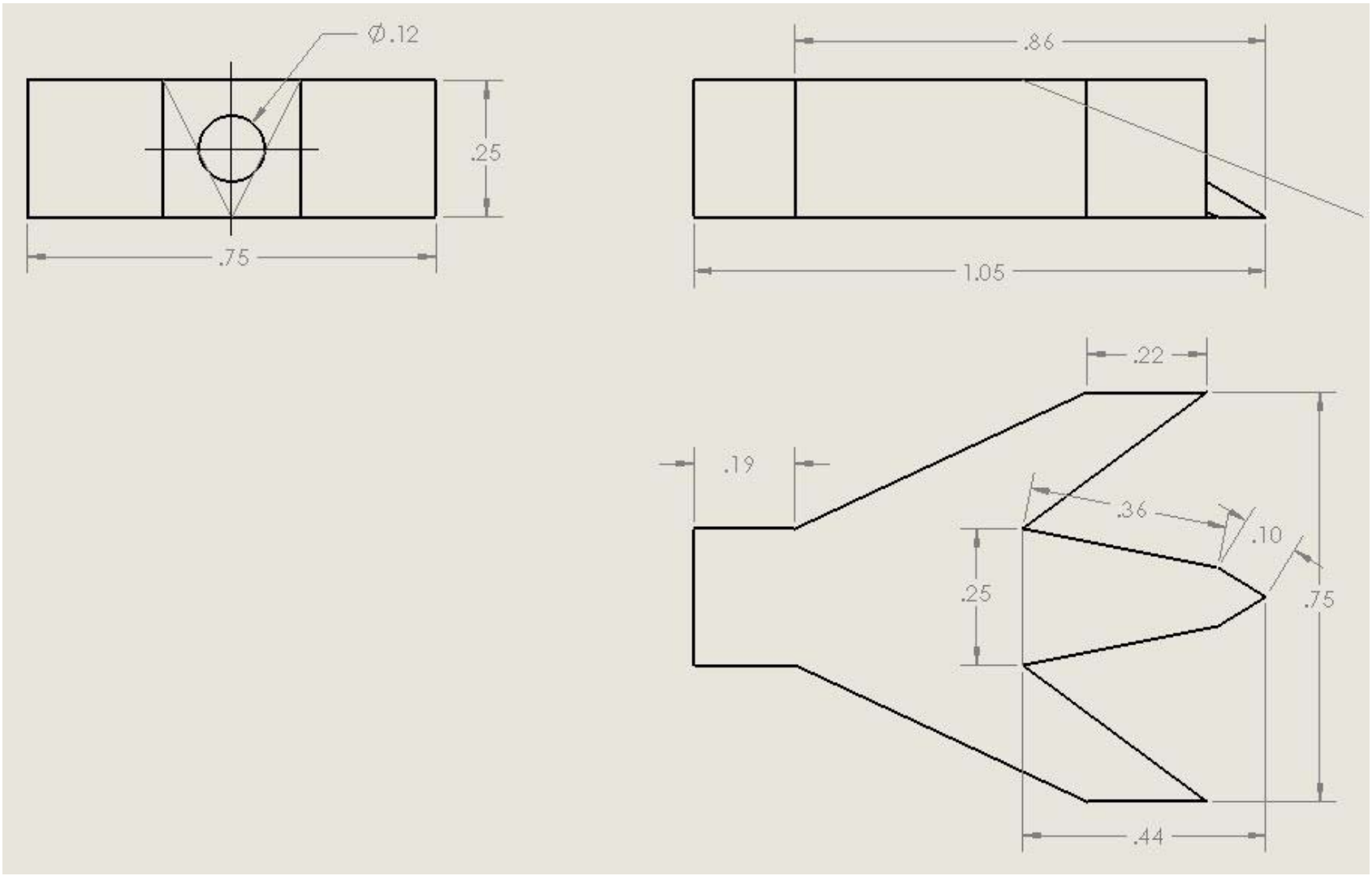

Figure 19. Detailed Drawing of the Hook Capture Mechanism.

While this system was adequate for the Pun-Jet Gliding ADS Prototype, as shown in Figure 14, the forces associated with the pneumatic launch of the Arcturus T-20 UAV and the subsequent force placed on the release mechanism on the Sparrow Gliding ADS Prototype were too great and resulted in multiple premature detachments. Therefore, two alternative methods of deploying the Sparrow Gliding ADS Prototype were devised. The first method is shown in Figure 15; the mounting bracket that was designed to mounting into the receptacle under either wing of the Arcturus T-20 UAV was repurposed and taped to the bottom of the fuselage of the Inspire 1 quad-copter. While this design was adequate, it did present three main issues:

1. In order for the mounting bracket to lay flush to the bottom of the Inspire 1 quad-copter, the flight camera had to be removed and the sensors used to determine the distance to the ground during the landing sequence were covered. Therefore, the Inspire had to be flown manually for the duration of each flight.

2. Clearance issues between the wings of the Sparrow Gliding ADS Prototype and the feet of the Inspire while mounted prevented launching the unit from the ground. Therefore, two pelican cases were placed on either side of the Inspire to ensure the proper clearance during launch.

3. The uneven wash off of the propellers of the Inspire caused the Sparrow airframe to twist on the mounting plate (shown in Figure 15).

Therefore, the second method, shown in Figure 20, was devised. A piece of line was attached to each of the four feet of the Inspire 1. These lines were then tied together at an equidistant point from each foot. A fifth line with a large loop on the end was attached to the knot where the other four were tied together. The loop was then fed through the slot in the bottom of the Sparrow Gliding ADS Prototype in the same way that was utilized in the alternative method. A golf ball was then taped to the line to add weight in order to ensure that the line did not rise into the propellers during the post-launch descent of the Inspire 1. This method prevented the issues mentioned above and was, therefore, used for a majority of the Sparrow Gliding ADS Prototype flights.

Figure 20. Inspire 1 Hanging Deployment Method for Sparrow Gliding ADS

D. AUTOMATED GUIDANCE UNIT

The Pun-Jet and Sparrow Gliding ADS Prototype Automated Guidance Units (AGU) consisted of the following parts:

- 3D Robotics (3DR) PIXHAWK Flight Controller
- 3DR Ublox GPS and Magnetometer Unit
- 3DR 900MHz Wireless Telemetry Modem Suite
- Spektrum DX7 Radio Transceiver and corresponding 2.4GHz Radio
- Turnigy 500mAh/7.4V 2S Lithium Battery
- 5A/5V Battery Eliminator Circuit
- Three (3) Turnigy TGY-50090M Analog Servos
- APM Planner 2.0 Open-Source Software Suite

These parts were connected as shown in Figure 21.

The PIXHAWK flight controller, formally produced by the 3D Robotics Company, is a low cost, commercially available automated guidance unit based on a 168MHz/252MIPS Cortex-M4F processor and designed to be compatible with multiple types of platforms including traditional fixed-wing aircraft, copters with multiple blades/propellers, and land-based roving vehicles. It contains a three-axis accelerometer, a three-axis magnetometer, an integrated barometer for detecting altitude, integrated backup systems for in-flight recovery and manual override with dedicated processor and stand-alone power supply, and dedicated terminals for a variety of peripherals including telemetry radios, external GPS and magnetometer sensors, a power control and sensor module, radio control units, and 14 pulse-width modulated (PWM), high-power compatible servo outputs. It is also capable of high data rate logging using a microSD card (PIXHAWK 2016). Because this unit costs only $199.99 new, it was an acceptable choice for this project and would likely be an adequate choice for a COTS-driven PADS.

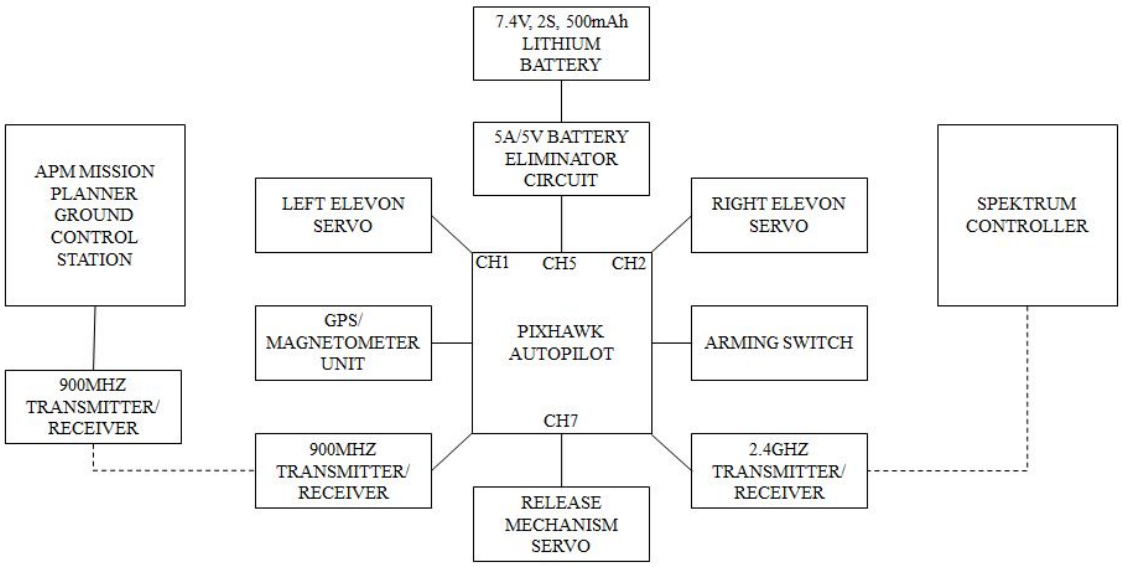

Figure 21. Wiring Diagram for Pun-Jet Gliding ADS Prototype

The 3DR Ublox GPS unit is an updated version of the Ublox LEA-6H module with a built-in HMC5883L digital compass. Typically, it is mounted away from the confines of the vehicle in order to decrease the interference seen in the compass readings. Since there is limited space inside both the Pun-Jet and Sparrow prototypes, the compass inside the Ublox GPS unit was disconnected and was used only for GPS readings while the compass internal to the PIXHAWK was used to determine bearings. Both the PIXHAWK and Ublox GPS unit are shown in Figure 22.

Each prototype was connected by a set of 915MHz telemetry radios to a laptop running APM Planner 2.0, an open-source ground control station (GCS) application that can be run on Windows, Mac or Linux. It features customized software for different airframe types that can be loaded onto any MAVlink-based autopilot. It can be used to plan missions with GPS waypoints and control events as well as view live data and issue commands, including changes to specific parameters and flight modes, to an airborne system.

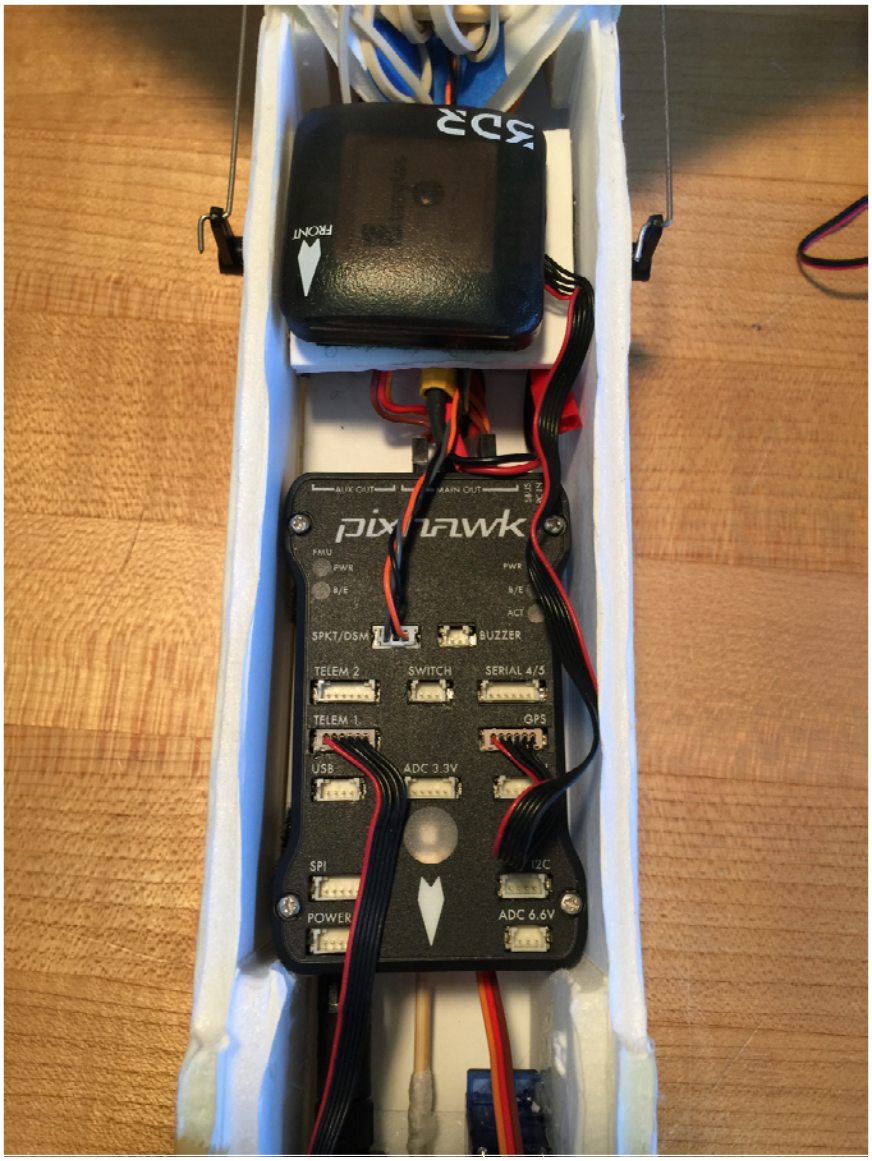

Figure 22. PIXHAWK Flight Controller and Ublox GPS Unit Installed in Sparrow Gliding ADS Prototype

E. COST ANALYSIS

As discussed in Chapter II, parachute-based PADS that are currently fielded by the United States military are expensive. Only medium or heavier weight systems, or those exceeding 10,000-pound payload capacity, can possibly satisfy the metric of six dollars per pound of payload delivered required by the MCWL. The smallest parachute-based systems, or those with a payload capacity under 500 pounds, exceed this

requirement. When looking at the cost breakdown of the approximately $27,000 per unit price for these systems, approximately $15,000 of it is the ram-air parafoil and $12,000 is the airborne guidance unit (AGU) including ruggedized onboard computer, sensors, and servo actuators (Yakimenko 2016). As this research suggests, utilizing gliders as opposed to expensive parafoils and COTS components for the AGU should decrease ADS cost drastically.

Figure 23 demonstrates the cost estimate analysis for the gliding AADS capable of carrying 500lbs and less. It is anticipated that COTS autopilots and sensors, like the PIXHAWK and 3DR UBlox GPS used in this project, should be able to serve as a core of the AGU for this entire weight range. While smaller systems may utilize standard RC servos to control the glider surfaces, it is estimated that slightly larger servos would be required for the systems whose payloads exceed 150 pounds. Also, while the glider airframe can be constructed out of foam, as demonstrated in this project, or plywood, larger systems would likely require stronger materials to be used.

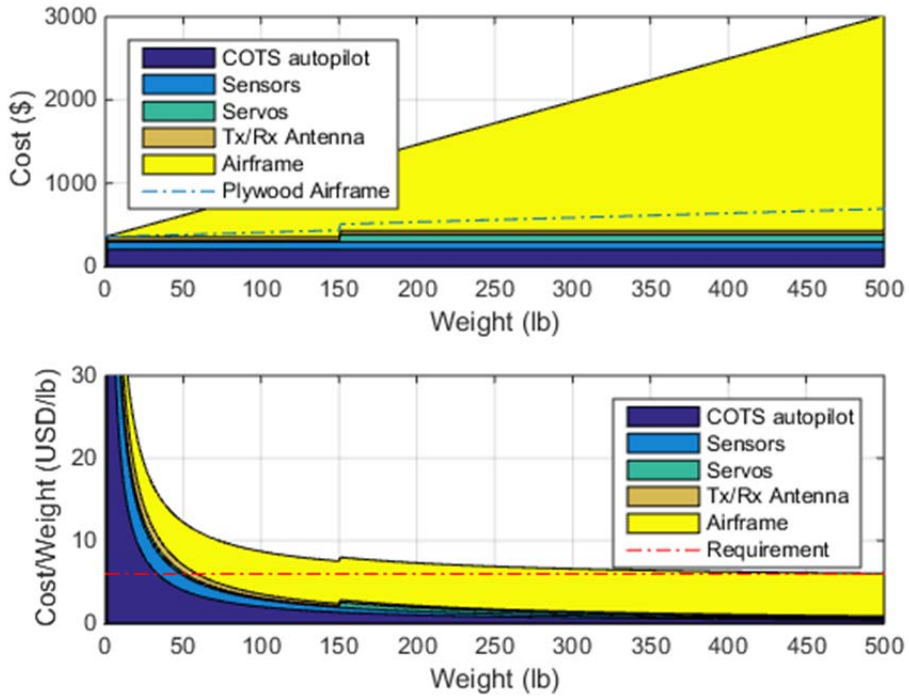

Figure 23. Gliding ADS Prototype Cost and Cost per Payload Weight Analysis

The aforementioned assumptions, and the fact that different-weight systems will have the same wing loading, were used to develop the estimates in Figure 23. In the upper portion of the figure, the dotted line shows the cost of the gliding ADS with plywood wings and the upper limit shows the cost of the gliding ADS with steel wings of the required surface area. The dotted line in the lower portion of Figure 23 shows the requirement of $6 per pound of payload delivered, three to four times better than the analogous weight-capability parafoil-based system, as set by the MCWL. While the gliding ADS meets the requirement at a payload of 500 pounds, the system exceeds the requirement at lower payload weights due to increased airframe costs. The cost of the AGU, however, stays relatively fixed with the only increased cost being the servos. The lower bound shown in the lower portion of Figure 23 could help making design decisions for the future smaller-weight AADS.

F. FLIGHT AND LANDING ALGORITHM

A critical part of any prototype UAV is the flight and landing algorithm. For this project, the following simple flight and landing algorithm was developed in order to test the ability of the COTS ADS electronics to support customizable software. The generic flight and landing algorithm equation is:

$$t_{total} = t_{phase} + t_{orbit} = \frac{(\theta_1 - \theta_0) r}{V_{TAS}} + \frac{2\pi r n}{V_{TAS}} \quad (1)$$

where t_{total} is the total flight time, r is the radius of the orbit, V_{TAS} is the true airspeed, $\theta_1 - \theta_0$ is the phase angle and n is the total number of orbits required to reach the landing point.

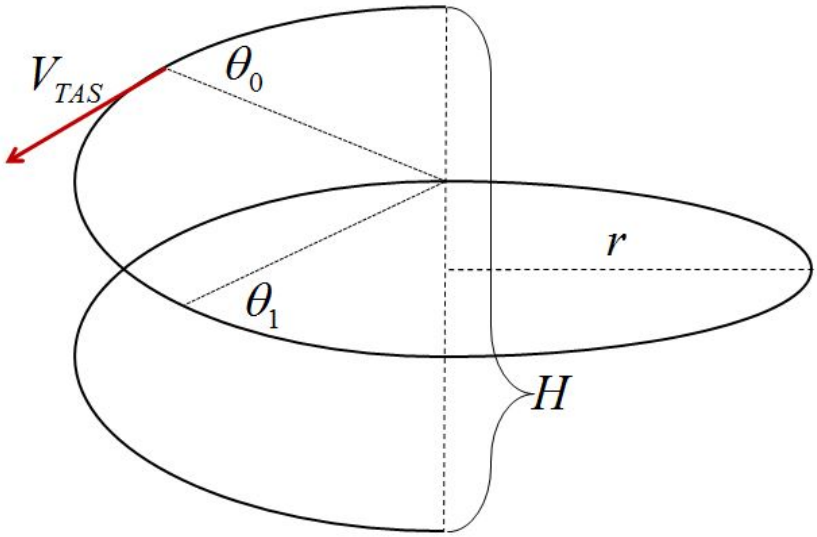

Figure 24. Flight and Landing Algorithm Pictorial Description

The total time flight time can also be described by

$$t_{total} = \frac{H}{\dot{h}} \tag{2}$$

where H is the total height above ground level and 180 degrees out of phase with the required landing heading when the orbit starts and $\dot{h}$ is the rate of descent. Substituting Equation 2 into Equation 1 leads to the following:

$$\frac{HV_{TAS}}{\dot{h}r} = \left(\theta_1 - \theta_0\right) + 2\pi n \tag{3}$$

Rearranging Equation 3 leads to a more useful form:

$$n = \frac{\frac{HV_{TAS}}{\dot{h}r} - \left(\theta_1 + \theta_0\right)}{2\pi} \tag{4}$$

Therefore, the number of orbits can be determined using Equation 4 and the time-dependent data. Rounding this number and substituting it back into Equation 4 yields, assuming all other values are left the same, the orbital radius that the airframe must be able to adjust to and hold in order to landing in phase with the runway.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. FIELD TESTS

All of the experimental test flights were conducted at McMillan Airfield on Camp Roberts California National Guard Base located approximately 15 miles north of Paso Robles, California. The airfield, which includes a 3,500-foot runway, is maintained and operated by the Naval Postgraduate School organization Center for Interdisciplinary Remotely-Piloted Aircraft Studies (CIRPAS). The mission of CIRPAS is to "provide a base of operations for UAV flight activities for internal and customer aircraft from the military, scientific, and developmental arenas" (CIRPAS 2016a). The airspace above and surrounding McMillan Airfield is restricted and, therefore, is controlled by Federal Aviation Administration (FAA) regulations. The Federal Aviation Administration (FAA) cautions that "restricted areas are established to separate activities considered to be hazardous to other aircraft" (CIRPAS 2016b). Therefore, the restricted area above McMillan Airfield "can be activated by CIRPAS in coordination with the appropriate base enabling UAV flights without interference with, or to, other aircraft" (CIRPAS 2016a). Additionally, the restricted area is "surrounded by a series of Military Operating Areas (MOAs) to help facilitate Air Operations" (CIRPAS 2016a). Figure 25 shows the airspace around Camp Roberts, CA, and McMillan Airfield.

The McMillan Airfield runway sits in a valley. Rapidly shifting wind conditions at altitudes lower than 1,000 meters AGL are caused by thermal upwelling from temperature deviations between air columns and updrafts caused by air being forced over small foothills approximately 100 meters north of the runway and a small mountain chain approximately two kilometers to the south and west of the airfield. These variable wind conditions can negatively affect small UAV operations (O'Brian 2016) and account for some of the instability in the data presented in sections A and B below.

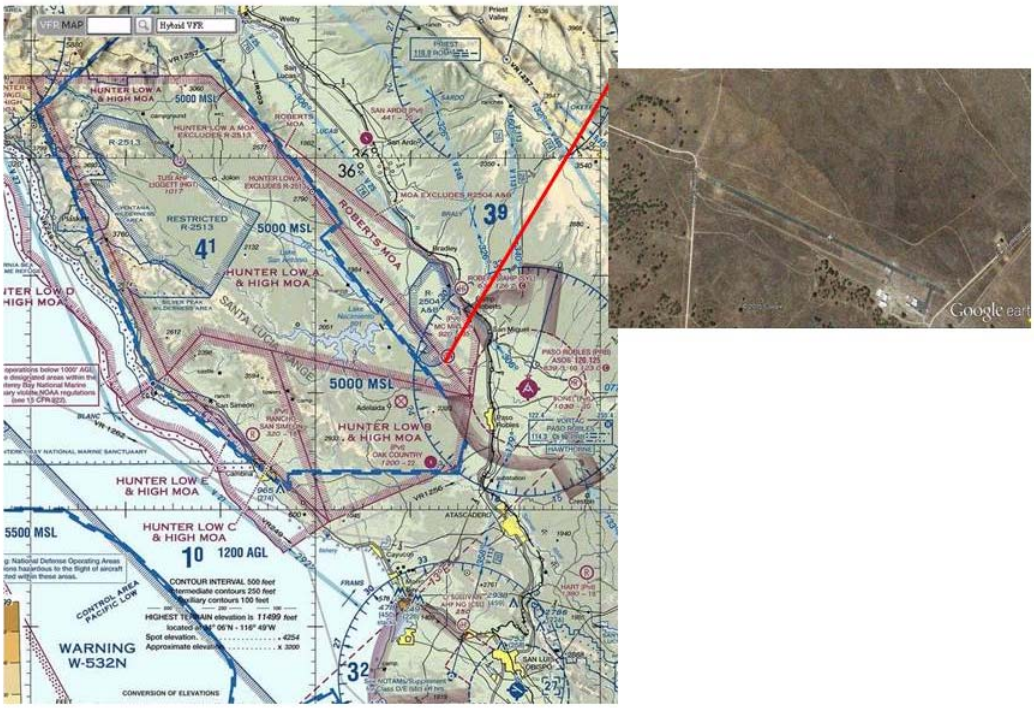

Figure 25. Visual Flight Rules Sectional Chart of the Camp Roberts Airfield (left) and an Aerial Photo of the McMillan Airfield Layout (right). Sources: O'Brian (2016).

A. PUN-JET GLIDING ADS PROTOTYPE TESTING AND ANALYSIS

Experimental flights for the Pun-Jet Gliding ADS Prototype were conducted in December of 2015 at the McMillan Airfield facility on Camp Roberts, CA. Each flight was conducted in the following sequence:

1. Battery was attached to the UBEC and power to all components was verified.

2. Communications between the ADS and GCS were verified.

3. Manual control and is verified by moving all servos through the full range of PWM values. This also ensures that there are no binding issues with any of the servos.

4. "Ready to Arm," indicated by a slowly flashing green LED on the front of the PIXHAWK, was verified. The arming sequence was subsequently initiated on the GCS and "armed" was verified on the PIXHAWK as indicated by a solid green LED.

5. Fuselage opening was closed and secured using tape.

6. Airframe was attached to the Arcturus T-20 UAV.

7. The pneumatic launch sequence was conducted by the delivery vehicle.

8. When the delivery vehicle has reached deployment altitude, the deployment sequence is initiated by channel 7 on the transmitter.

9. Following touchdown, the airframe was retrieved and powered off. The data flash log was pulled from the micro-SD card in the PIXHAWK and analyzed using the Pun-Jet Gliding ADS Prototype Analysis Script in MATLAB as shown in Appendix A.

Of the seven experimental flights that were conducted, only one was fully autonomous. The initial data and analysis is from that fully autonomous flight (Flight 6) and was analyzed in coordination with Lieutenant Ryan Beall, USN. The data from this flight was shaped using two triggers; first, the time of deployment was found using the channel 7 PWM decrease when the airframe was at altitude and, second, the spike in acceleration sensed by the IMU when the aircraft impacted the ground. These triggers in the form of MATLAB code are shown in Figure 26.

```
%%
acceleration = sqrt(IMU2(:,6).^2+IMU2(:,7).^2+IMU2(:,7).^2);
[Index_drop,~]=find(RCOU(:,9)<1300 & BARO(:,3)>100);
Index_drop = min(Index_drop);
a_total = interp1(IMU2(:,2),acceleration,RCOU(:,2),'pchip');
[Index_land,~]=find(a_total>20);
Index_land = max(Index_land);
time_of_release = RCOU(Index_drop(1),2);
time_of_land = RCOU(max(Index_land),2);

flight_time = (time_of_land - time_of_release )/(1*10^6)/60;
```

Figure 26. MATLAB Code from Pun-Jet Analysis Script to Find Release and Landing Points. Source: Beall and Henderson (2016).

After the data was parsed, it was run through the remainder of the MATLAB script. Figures 27 and 28 illustrate a birds'-eye view and three-dimensional view of the flight path, respectively, and also demonstrate the basic attitude hold performance, of the airframe.

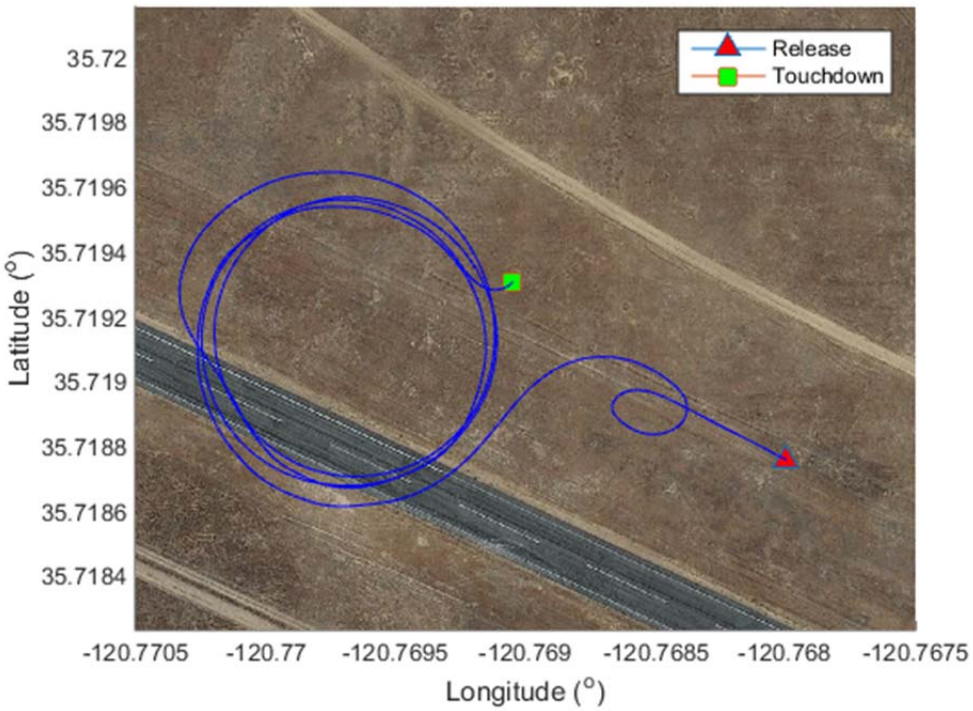

Figure 27. Pun-Jet Initial Flight: Birds'-Eye View Plot.

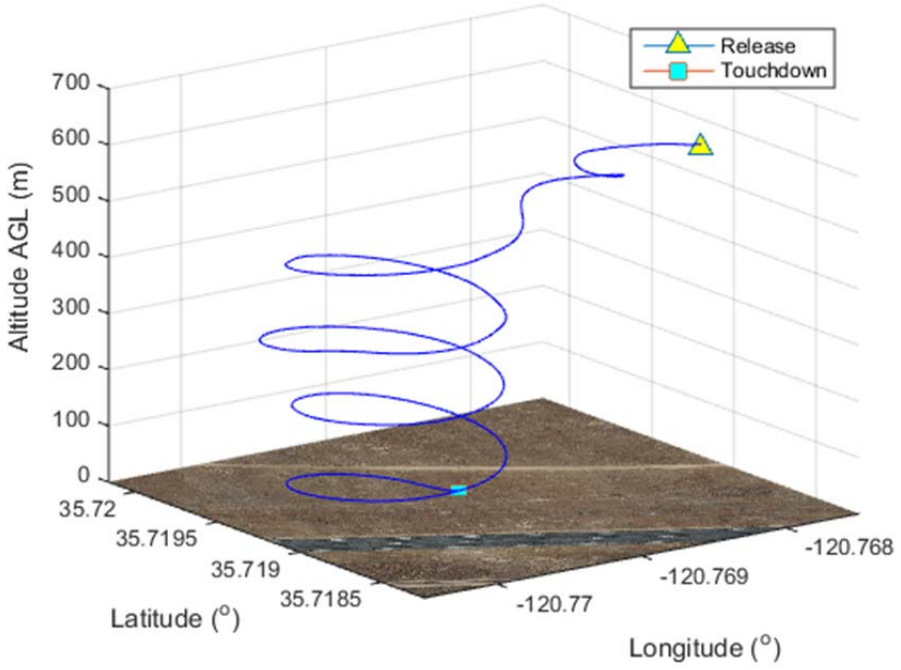

Figure 28. Pun-Jet Initial Flight: Three-Dimensional View of the Descent.

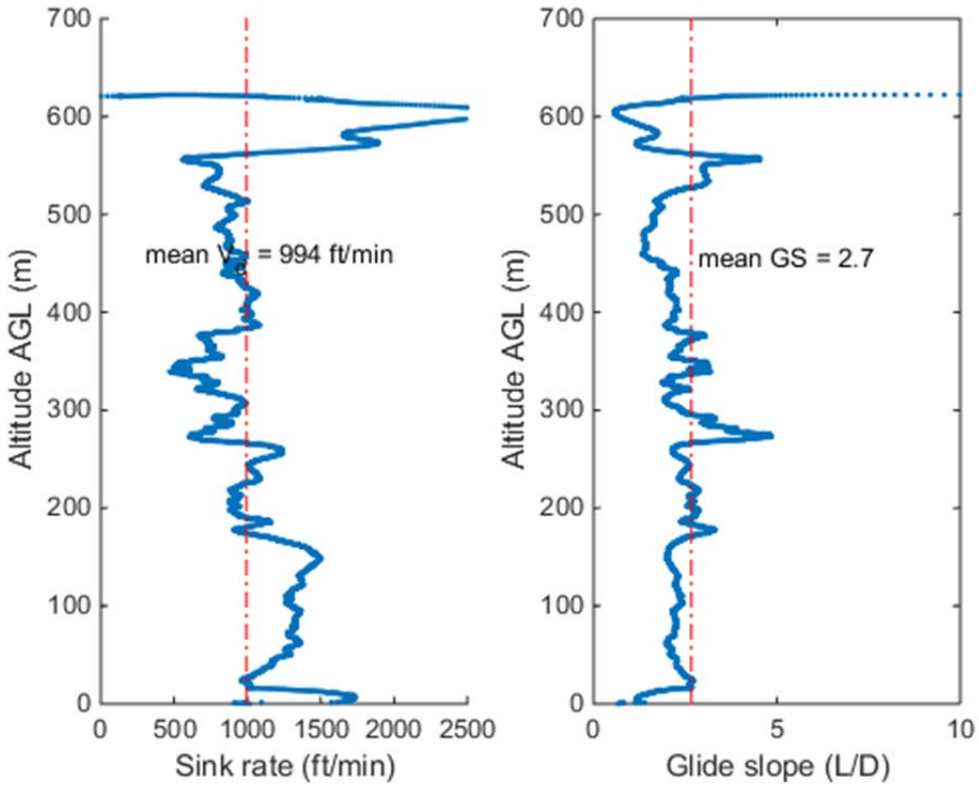

Figure 29. Pun-Jet Initial Flight: Sink Rate and Glideslope Analysis. Source: Beall and Henderson (2016).

Figure 29 shows the expected glide performance of the Pun-Jet airframe at its current wing loading. Both the sink rate and glide slope stay relatively fixed around the average of 995 feet per minute and 2.37:1, respectively, throughout the descent. This relatively low glide slope does not meet the threshold requirement of 5:1 set by the MCWL as described in Chapter III of this thesis. A desire to test an airframe that met these requirements led to the development of the Sparrow Gliding ADS Prototype whose analysis is described in the subsequent section of this thesis.

Figure 30 shows the roll attitude hold performance during the orbit over the airfield. While it is clear that the roll attitude hold PID loop is working, it is also evident that the PIDs are underperforming when introduced to large perturbations in roll attitude and the error tracking performance could be greatly improved with more tuning flights.

However, the flight profile analysis in Figures 27, 28, 29 and 30 demonstrated that system identification tools could be used to accurately model the flight.

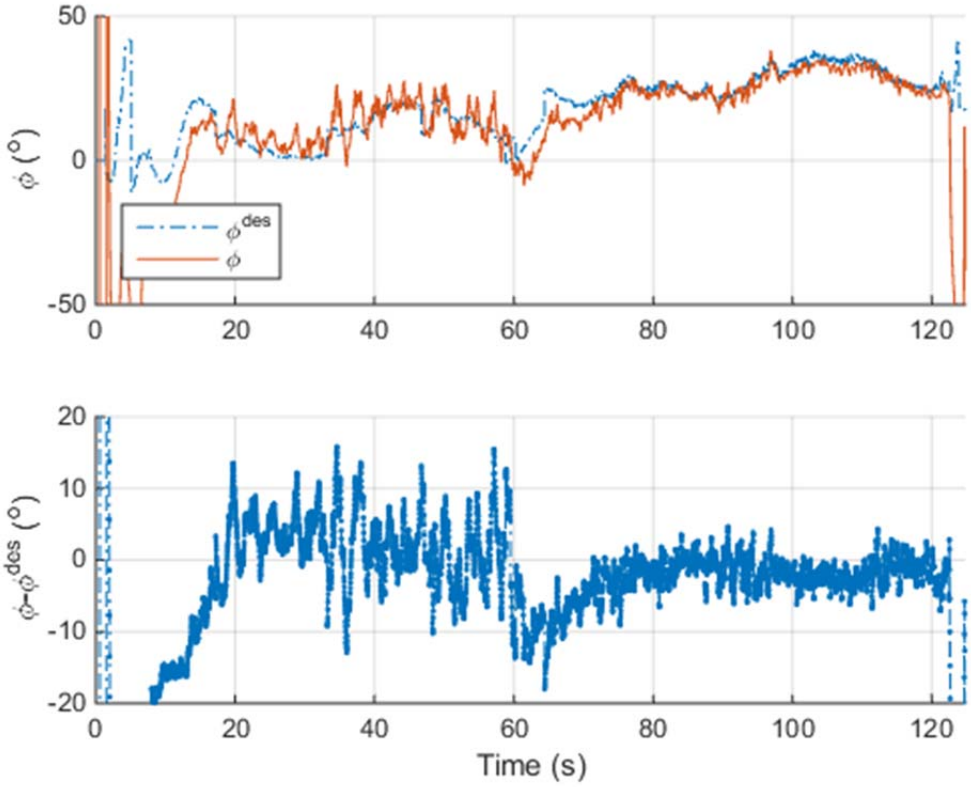

Figure 30. Pun-Jet Initial Flight: Roll Attitude Hold Performance. Source: Beall and Henderson (2016).

The Single Input, Single Output (SISO) MATLAB method was used to analyze the roll rate response versus the commanded roll in order to estimate the performance of the Pun-Jet airframe. The model for roll was determined to be

$$y = k\frac{a}{s+a} = 0.0213\left(\frac{35.06}{s+35.06}\right) \tag{5}$$

The roll command PWM values recorded from the flight were fed back into the model and compared to the actual roll rate achieved by the aircraft. Figure 31 shows that

the model inadequately mapped to the actual performance of the airframe. This error is likely due to noise in the data from natural unstable air disturbances as described in the introduction to this chapter. Therefore, additional analysis was required to further refine the model and better understand the System Identification and PID optimizations.

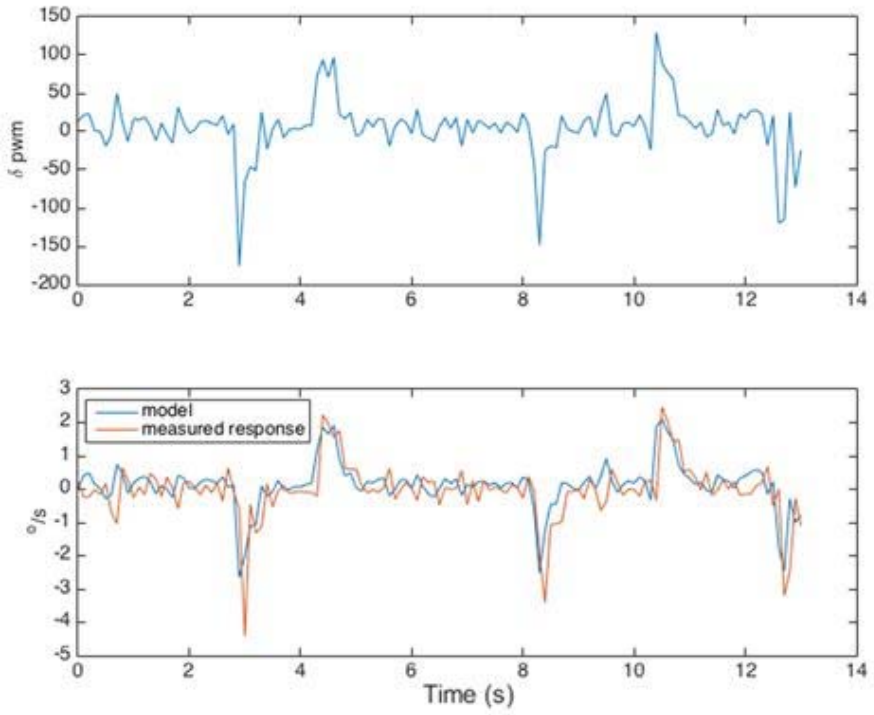

Figure 31. Roll Rate Model Validation. Source: Beall and Henderson (2016).

During the initial flight, the bank angle controller also performed suboptimally (Beall and Henderson 2016). As Beall and I noted, the gains were likely too high as evidenced by the two to three degree oscillations in amplitude with a period of one-half to one second. Therefore, a clean step response was found in the roll data as seen in the upper portion of Figure 32 in order to generate a more accurate, second-order model that also accounted for servo delay:

$$\frac{X(s)}{U(s)} = \frac{k\omega_n^{\,2}}{s^2 + 2\zeta\omega_n s + \omega_n^{\,2}} = e^{-0.04s}\left[\frac{0.0581(24.24)^2}{s^2 + 2(1.179)(24.24)s + (24.24)^2}\right] \tag{6}$$

The response of this model is shown in Figure 31. While it still is not perfect, it does represent a clear improvement to the first-order model and is slightly over-damped as shown in Figure 33.

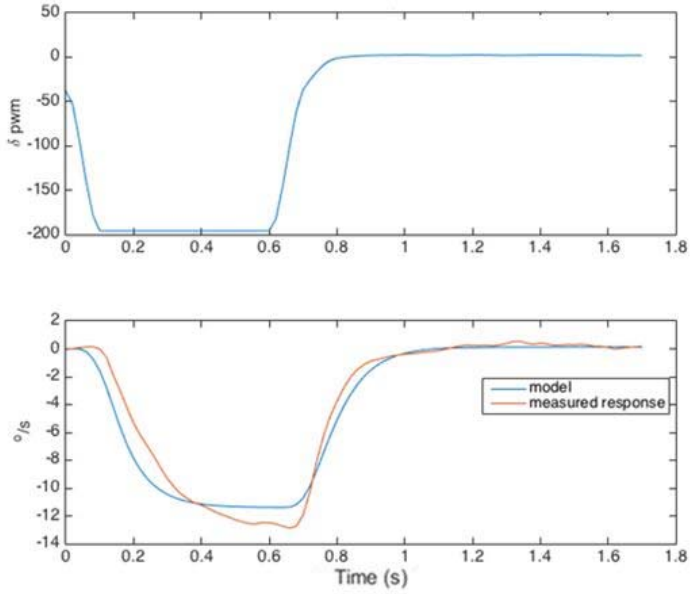

Figure 32. Improved Roll Rate Model Validation. Source: Beall and Henderson (2016).

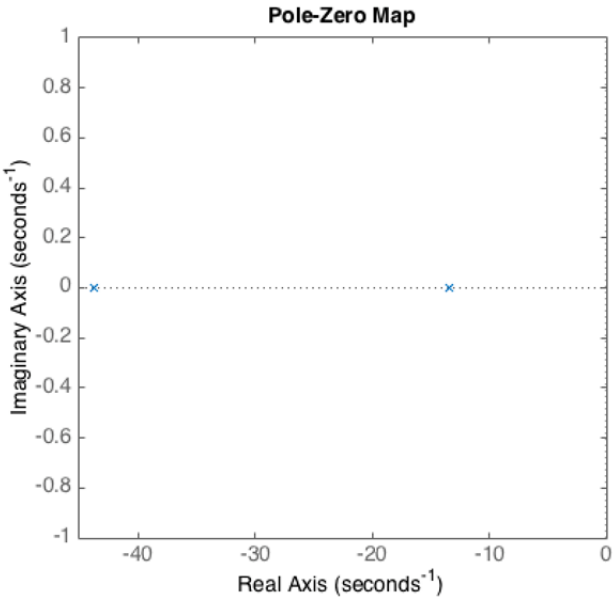

Figure 33. Pole-Zero Map. Source: Beall and Henderson (2016).

The model shown in Figure 34 was developed and implemented in Simulink to verify the airframe's performance with roll P, I, and D gains of 0.4, 0.05 and 0.05, respectively.

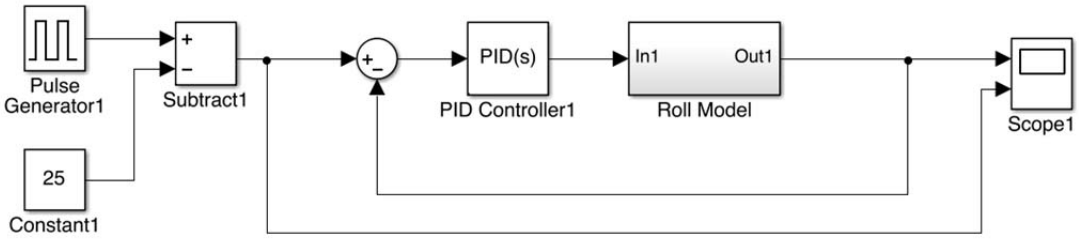

Figure 34. Flight PID Value Simulation Model.
Source: Beall and Henderson (2016).

Figure 35 shows the response for a ±25 degree roll step input (Beall and Henderson 2016). As Beall and I noted, this underdamped response is what was seen during the experimental flight and validates the model as a close approximation of the airframe.

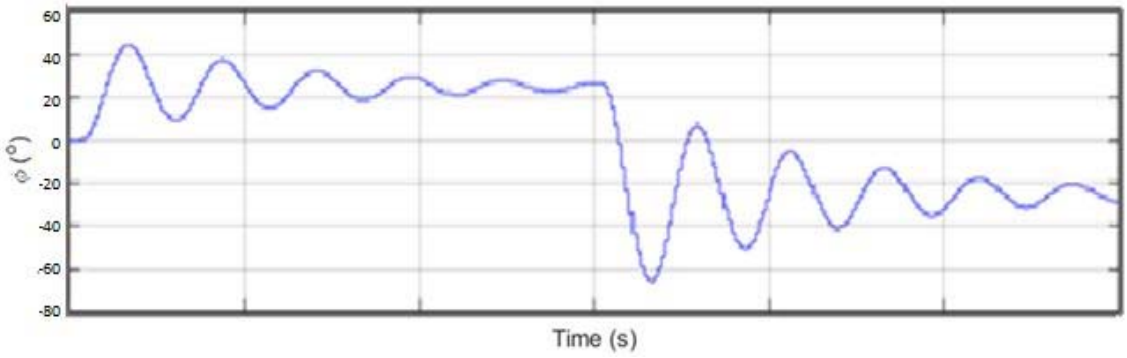

Figure 35. Expected Response with the Default PID Values.
Adapted from Beall and Henderson (2016).

The improved performance shown in Figure 36 was achieved by tuning the roll P, I, and D values to 0.13, 0.02 and 0.04 which represents a halving of the P gain and a slight decrease in the I and D gains.

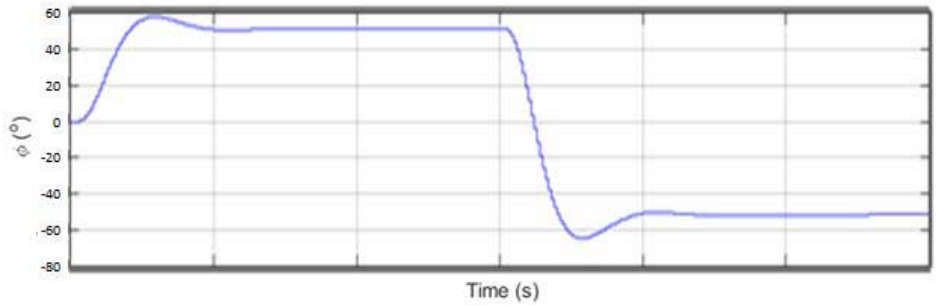

Figure 36. Expected Response with Updated PID Values

The second round of experimental flights were conducted on February 16 and 17 2016 using these updated PID values and a decreased L1 Navigation period of 12. The following analysis was conducted on one of these flights. Figures 37 and 38 show the birds'-eye view and three-dimensional view of the descent, respectively.

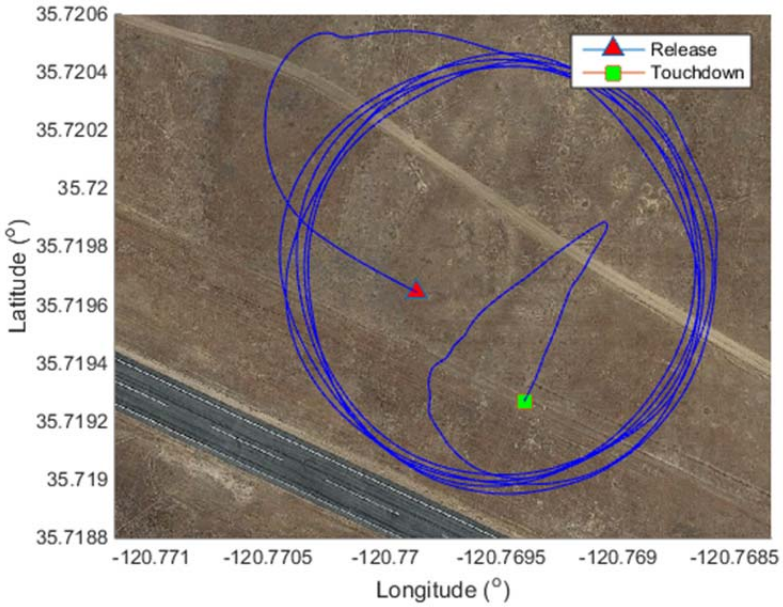

Figure 37. Pun-Jet Improved Flight: Birds'-Eye View of Descent

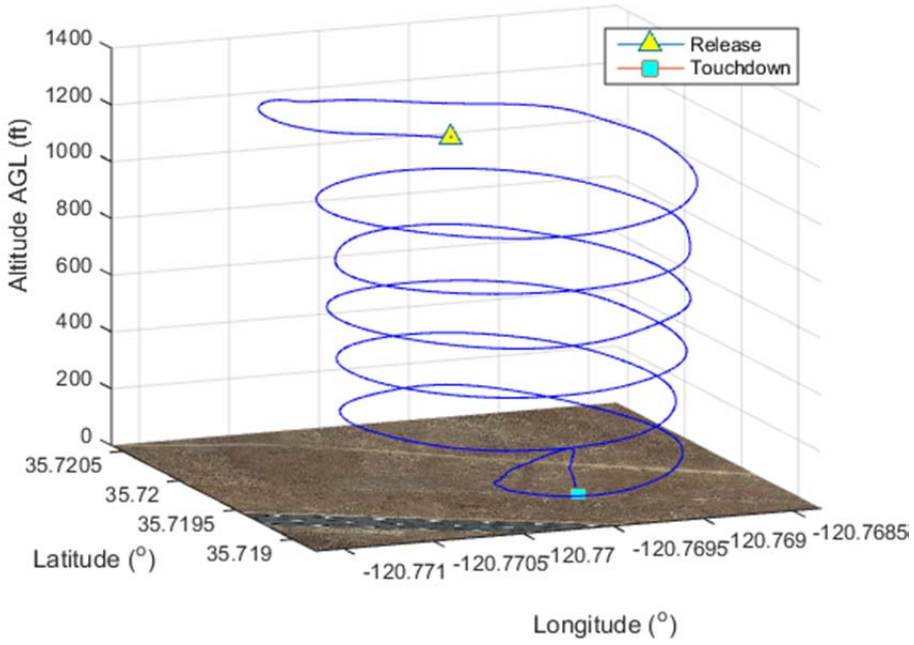

Figure 38. Pun-Jet Improved Flight: Three-Dimensional View of Descent

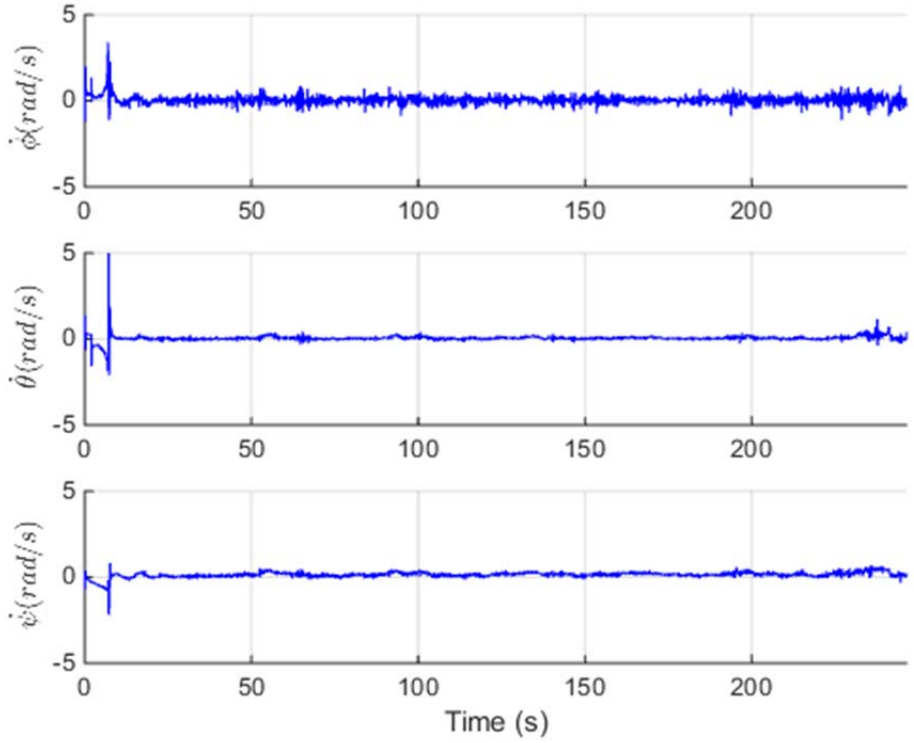

Figure 39. Pun-Jet Improved Flight: Roll, Pitch, and Yaw Rates

Figure 39 show the roll, pitch, and yaw rates for the airframe during the descent. After the change in the PID gains, the Pun-Jet is much more stable in roll with variations during the flight of less than one radian per second. These variations are likely attributed to variations in wind velocity. The increased stability in roll is evident in Figure 40 when comparing it to Figure 30. There is a marked increase in the performance of the roll attitude hold loop. The difference between the measured roll and the desired roll oscillates around zero, which is optimal and can likely be attributed to servo lag.

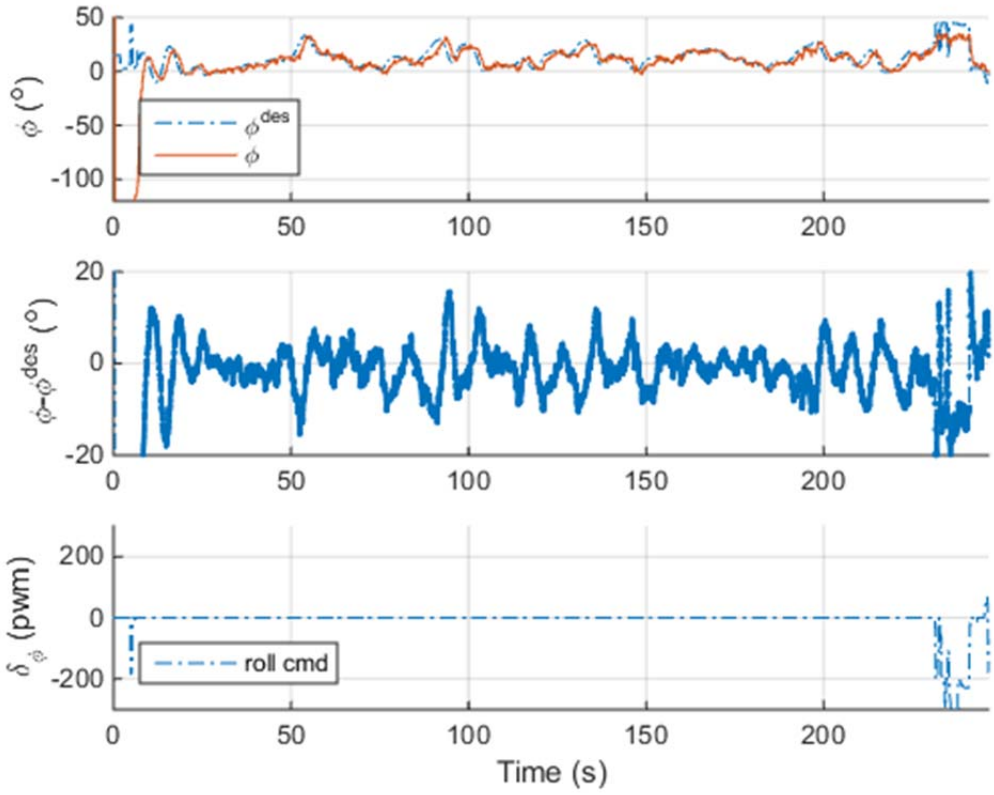

Figure 40. Pun-Jet Improved Flight: Roll, Desired Roll and Roll PWM Plots

Figure 41 shows the pitch, desired pitch and PWM values for the descent. It is evident that the pitch attitude hold loop is commanding a negative nose angle the entire flight in order to maintain airspeed and hold the commanded orbit radius.

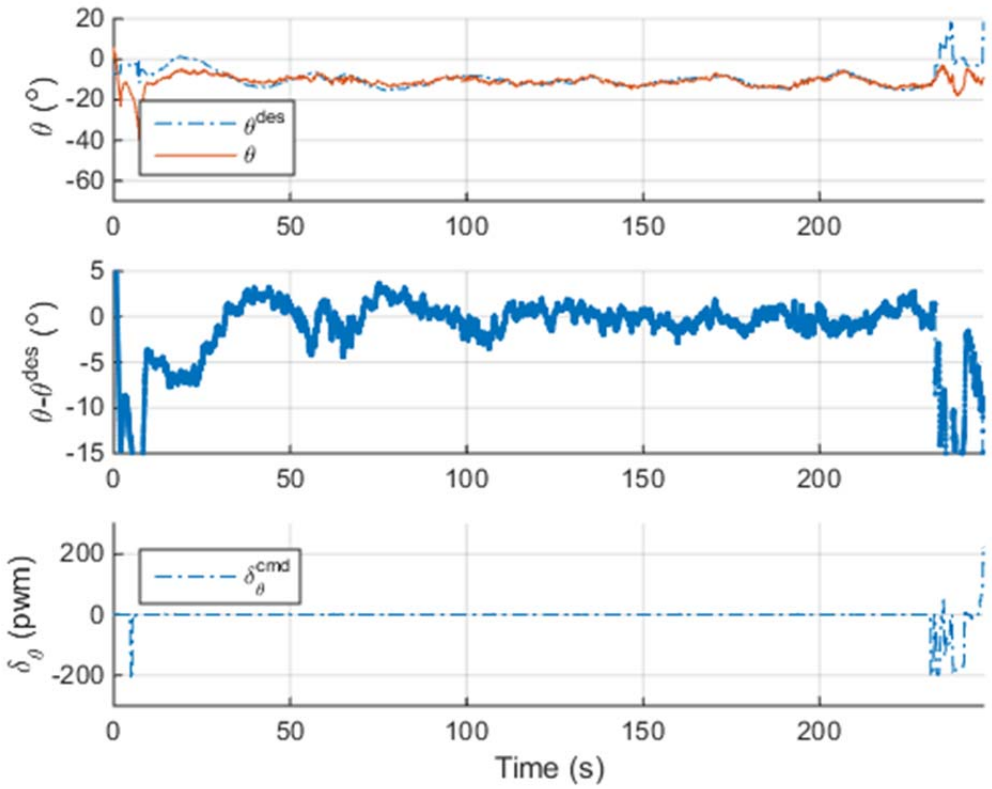

Figure 41. Pun-Jet Improved Flight: Pitch, Desired Pitch and Pitch PWM Plots

B. SPARROW GLIDING ADS PROTOTYPE TESTING AND ANALYSIS

Experimental flights for the Sparrow Gliding ADS Prototype were conducted on October 19–20, 2016 at the McMillan Airfield facility on Camp Roberts, CA. Each flight was conducted in the following sequence:

1. Battery was attached to the UBEC and power to all components was verified.

2. Communications between the ADS and GCS were verified.

3. Manual control and is verified by moving all servos through the full range of PWM values. This also ensures that there are no binding issues with any of the servos.

4. "Ready to Arm," indicated by a slowly flashing green LED on the front of the PIXHAWK, was verified. The arming sequence was subsequently initiated on the GCS and "armed" was verified on the PIXHAWK as indicated by a solid green LED.

5. Wing was attached using rubber bands.

6. Airframe was attached to the delivery vehicle.

7. Launch sequence initiated by the delivery vehicle.

8. When the delivery vehicle has reached deployment altitude, the deployment sequence is initiated by Channel 7 on the transmitter.

9. Following touchdown, the airframe was retrieved and powered off. The data flash log was pulled from the micro-SD card in the PIXHAWK and analyzed using the Sparrow Gliding ADS Prototype Analysis Script in MATLAB as shown in Appendix A.

The data from these flights were shaped using two triggers; first, the time of deployment was found using the Channel 7 PWM decrease when the airframe was at altitude and, second, the moment when the airframe descended below 1.5 meters AGL. Figure 41 shows the values of these triggers throughout the log and is presented before the data is parsed in the MATLAB script in order to provide an opportunity to adjust the values associated with the triggers if necessary.

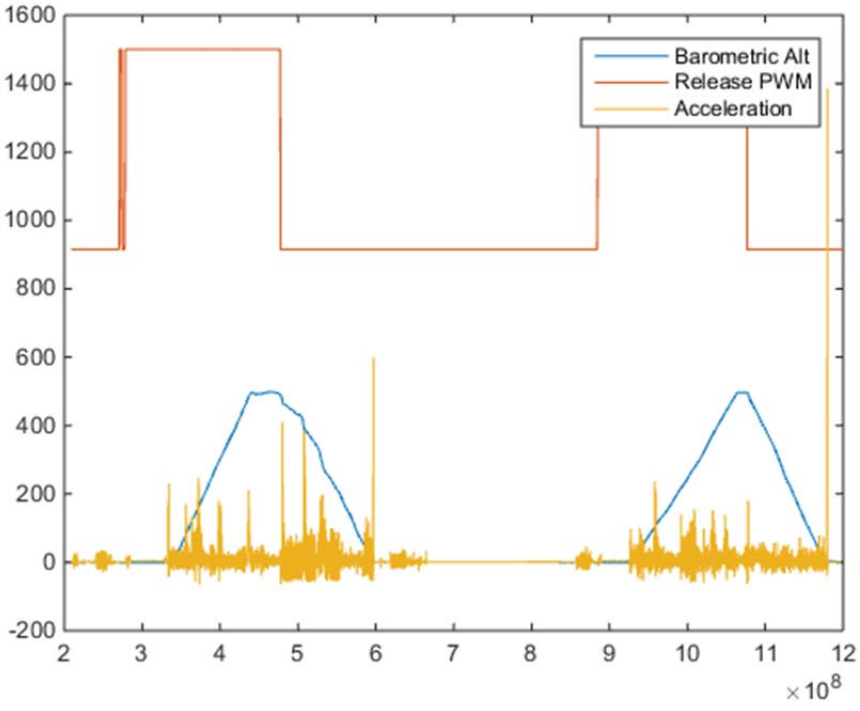

Figure 42. Sparrow Log Overview Plot

After the data was parsed, it was run through the remainder of the MATLAB script. The first flight on the log, hereafter referred to as the tuning flight, is analyzed below. Three flight modes were used during the Sparrow flights:

- Return to Launch (RTL): flight mode in which the plane will return to its home location and loiter, in an orbit, until given alternate instructions. This mode was used, along with the RTL_LOITER_RAD parameter, to determine the ability of the aircraft to maintain a commanded orbital radius.

- Manual: flight mode in which the plane is controlled by the RC transmitter by passing its commands through the PIXHAWK directly to the servos. This mode was used to correct uncontrolled flight and for landing when the approach taken by the autopilot was deemed unsuitable.

- Fly-By-Wire A (FBWA): flight mode in which the planes roll and pitch are constrained by the LIM_ROLL_CD, LIM_PITCH_MAX and LIM_PITCH_MIN parameter settings. This allows for controlled, assisted flight during initial tests and PID gain tuning.

Note that in all subsequent plots, colored lines or background shading are used to indicate which flight mode is being used during the flight with green representing the FBWA mode, red representing the RTL mode, and blue representing the Manual mode.

Figures 43 and 44 illustrate a birds'-eye view and three-dimensional view of the flight path, respectively, and also show the flight mode that the AGU was in during each phase of the flight. In order to save the effort of developing a model and to enable in-situ analysis of the flight characteristics for the Sparrow airframe, an alternative method was used to tune the PID gains using the plotting feature in MAVProxy, the Linux version of APM Mission Planner used during the Pun-Jet experimental flights. The FBWA flight mode was used while tuning the aircraft. Roll and pitch commands were introduced to the aircraft manually from the transmitter and the response to those commands, along with the original command, was plotted. The PID gains were adjusted using the "param set" command in MAVProxy. For instance, the roll P gain would be adjusted by typing "param set RLL2SRV_P" followed by the commanded value for that parameter.

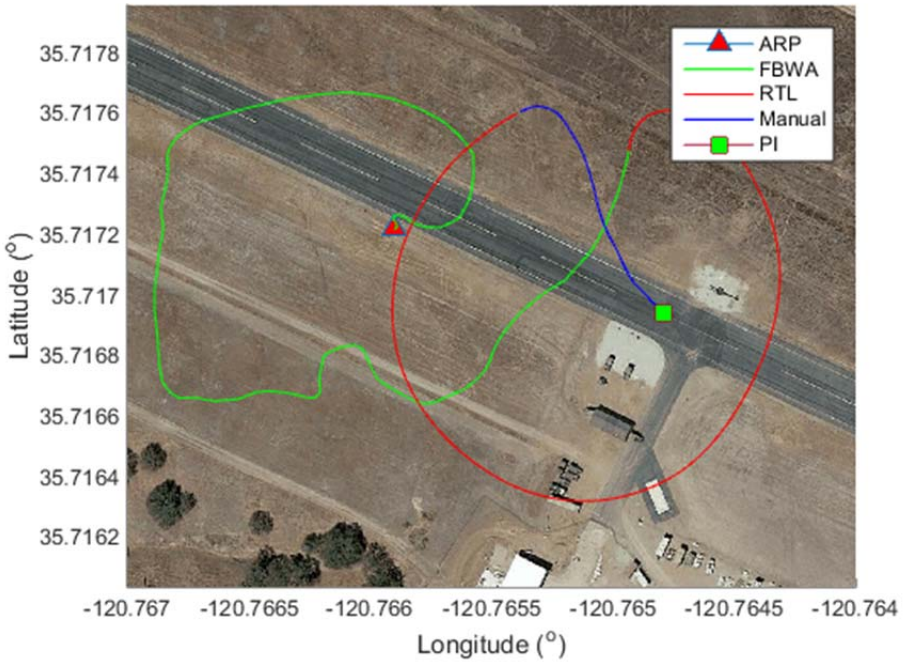

Figure 43. Sparrow Tuning Flight: Birds'-Eye View of Descent

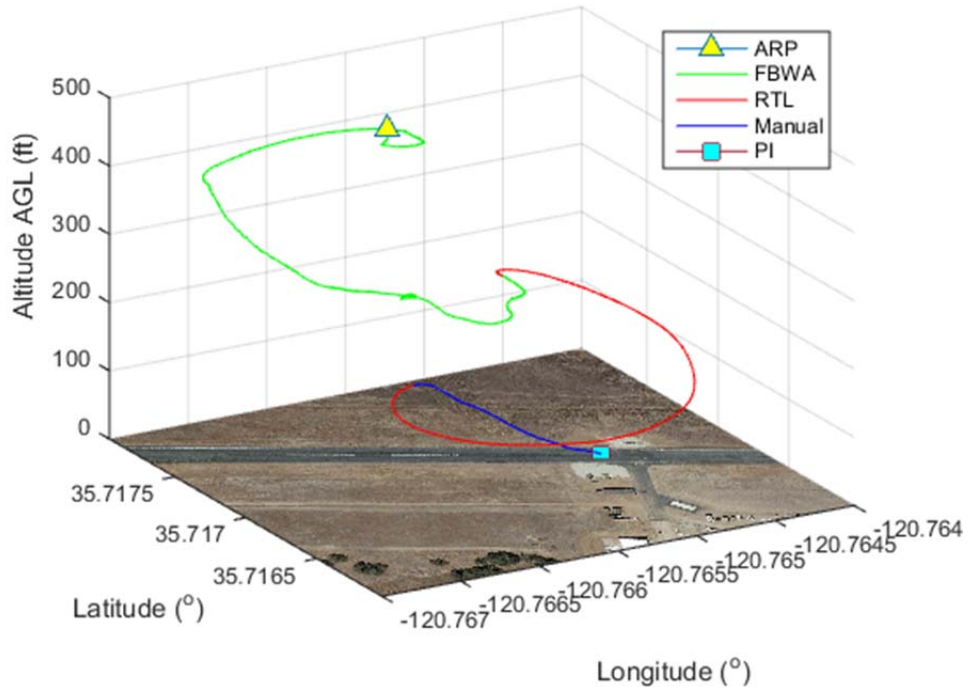

Figure 44. Sparrow Tuning Flight: Three-Dimensional View of Gliding Descent

Figure 45 shows the roll, pitch, and yaw rates for the tuning flight. The large oscillations in the roll, pitch, and yaw rates indicate that the aircraft is unstable. However, it is evident in the RTL portion of the flight that the gains have been adjusted appropriately. The larger oscillations in roll rate during the RTL portion of the flight are likely attributed to variations in wind velocity seen by the airframe during the descent. Likely causes for these variations in wind are described in the introduction to this chapter.

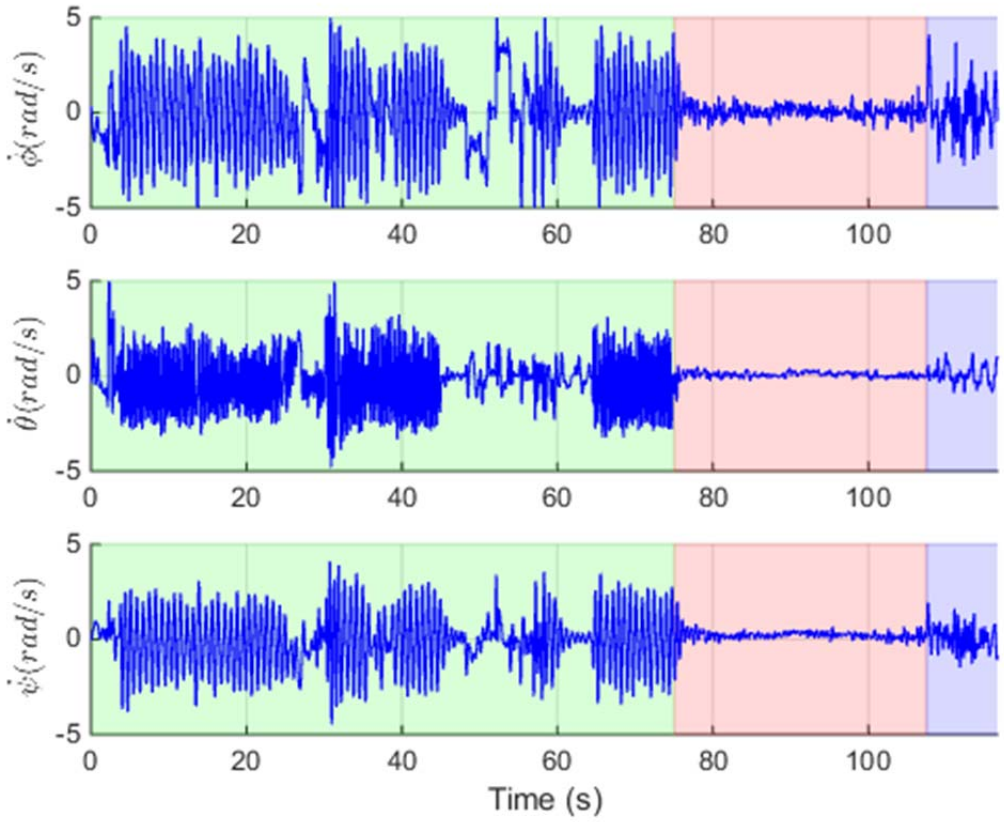

Figure 45. Sparrow Tuning Flight: Roll, Pitch, and Yaw Rates Versus Time

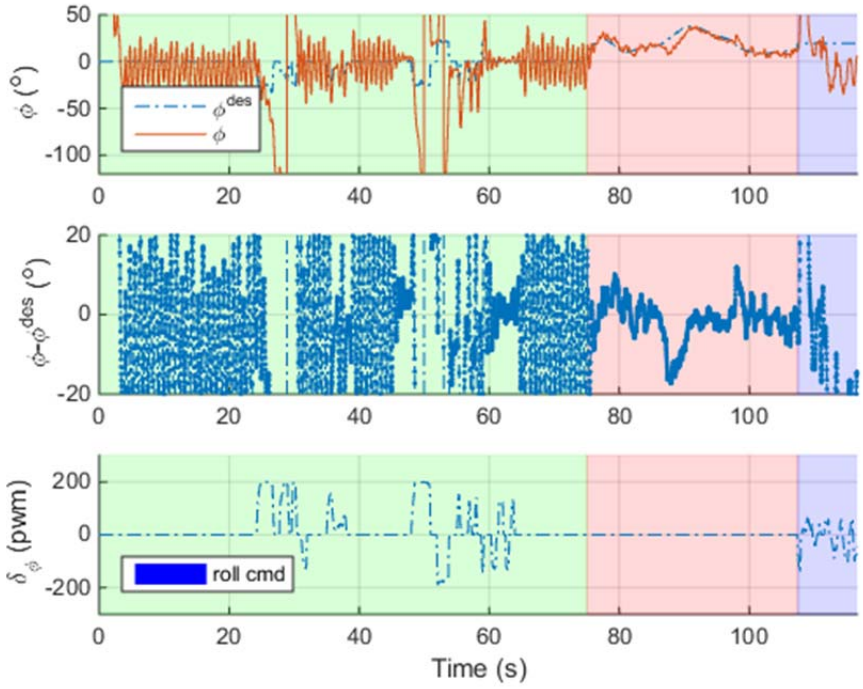

Figure 46. Sparrow Tuning Flight: Roll, Desired Roll and PWM Time Histories

Figures 46 and 47 show the roll, desired roll, pitch, and desired pitch. At approximately 25 seconds and again at 50 seconds into the flight, the airplane went into an uncontrollable state. Figure 48 shows the sink rate and glide slope ratio of the tuning flight at various altitudes. During the first period of uncontrollable flight, the plane dove at a 70 degree down angle with a 100 degree left angle of bank. The sink rate increased from approximately 400 feet per minute on average to approximately 2,500 feet per minute. The glide ratio of the Sparrow also decreased from an average of approximately 5:1 to less than 2:1. During the second, prolonged period of uncontrollable flight, the plane dove at a 50 degree down angle with an even more pronounced left angle of bank, the sink rate increased from approximately 500 feet per minute on average to approximately 1,750 feet per minute, and the glide ratio decreased from an average of approximately 5:1 to less than 2:1.

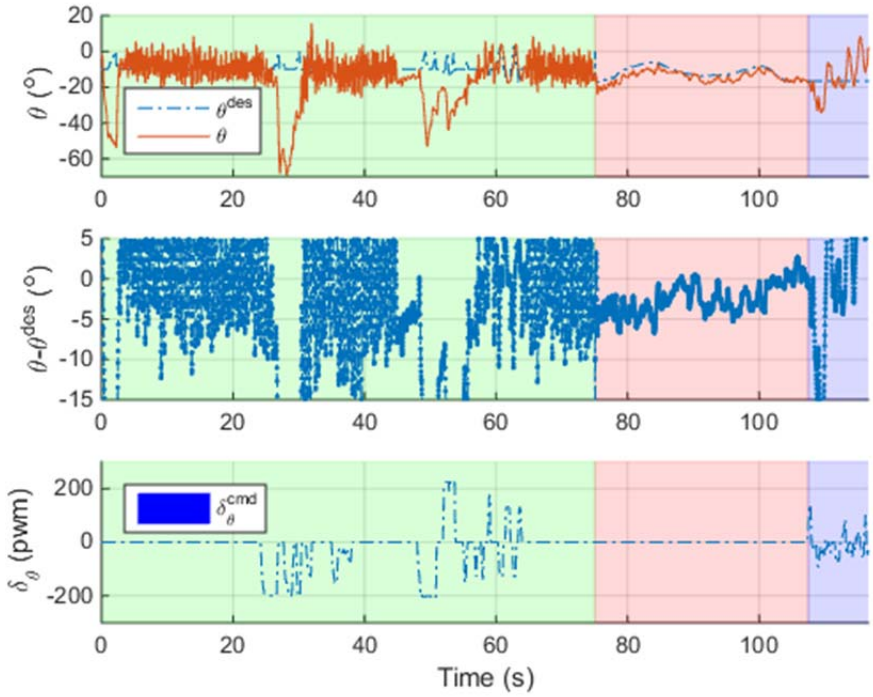

Figure 47. Sparrow Tuning Flight: Pitch, Desired Pitch and PWM Time Histories

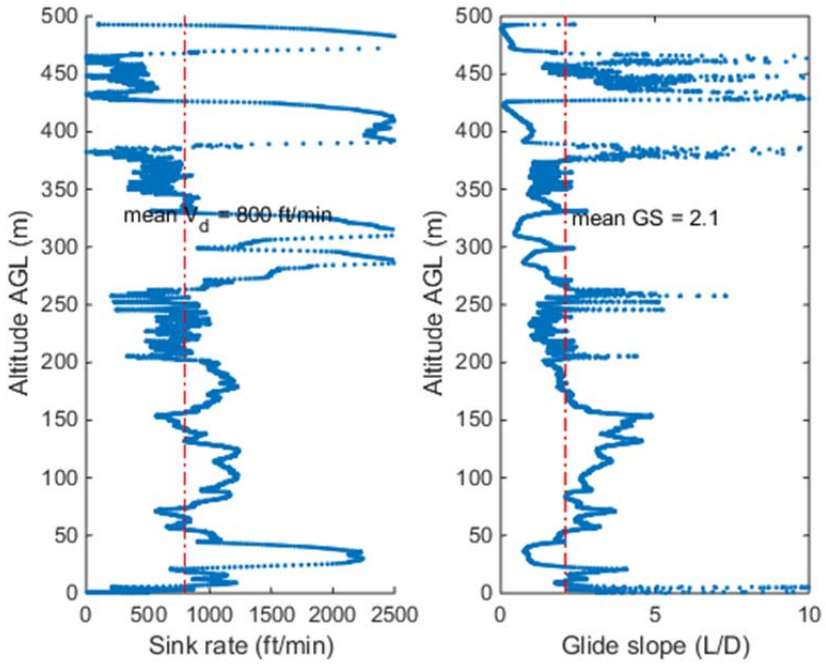

Figure 48. Sparrow Tuning Flight: Sink Rate and Glide Slope Versus Altitude

The uncontrollable state was attributed to tip stall caused by the rapid deceleration of the airframe as seen Figure 49. While the PIXHAWK does log an approximate airspeed, this data is not logged for all flight modes and was determined to be unreliable when logged. Therefore, the following equations were used to calculate the airspeed and the acceleration of the airspeed, respectively:

$$V_{as} = \sqrt{\left(V_n - V_{w,n}\right)^2 + \left(V_e - V_{w,\mathrm{e}}\right)^2} \tag{7}$$

$$\dot{V}_{as} = \int V_{as}\, dt \tag{8}$$

where V_n is the ground velocity in the north direction, $V_{w,n}$ is the wind velocity in the north direction, V_e is the ground velocity in the east direction, and $V_{w,\mathrm{e}}$ is the wind velocity in the east direction. The descent velocity increases drastically only after a large decrease in the airspeed of the plane.

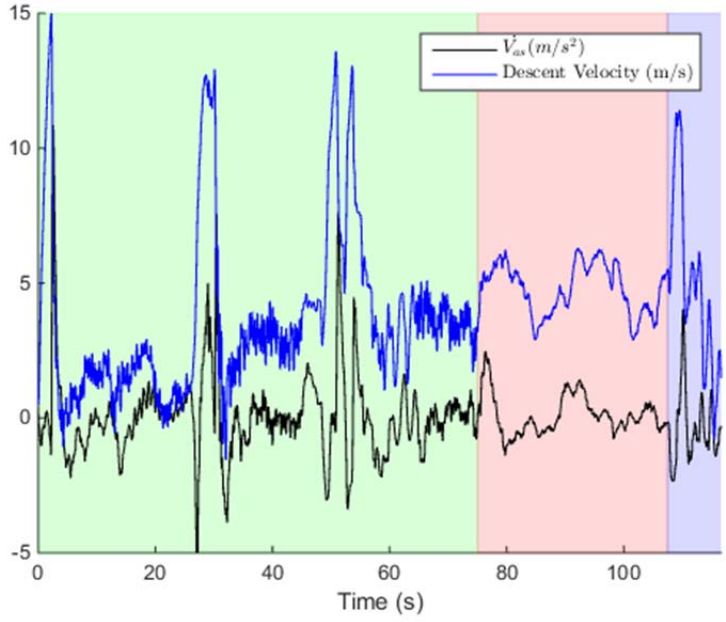

Figure 49. Airspeed Acceleration and Descent Velocity Correlation Time Histories

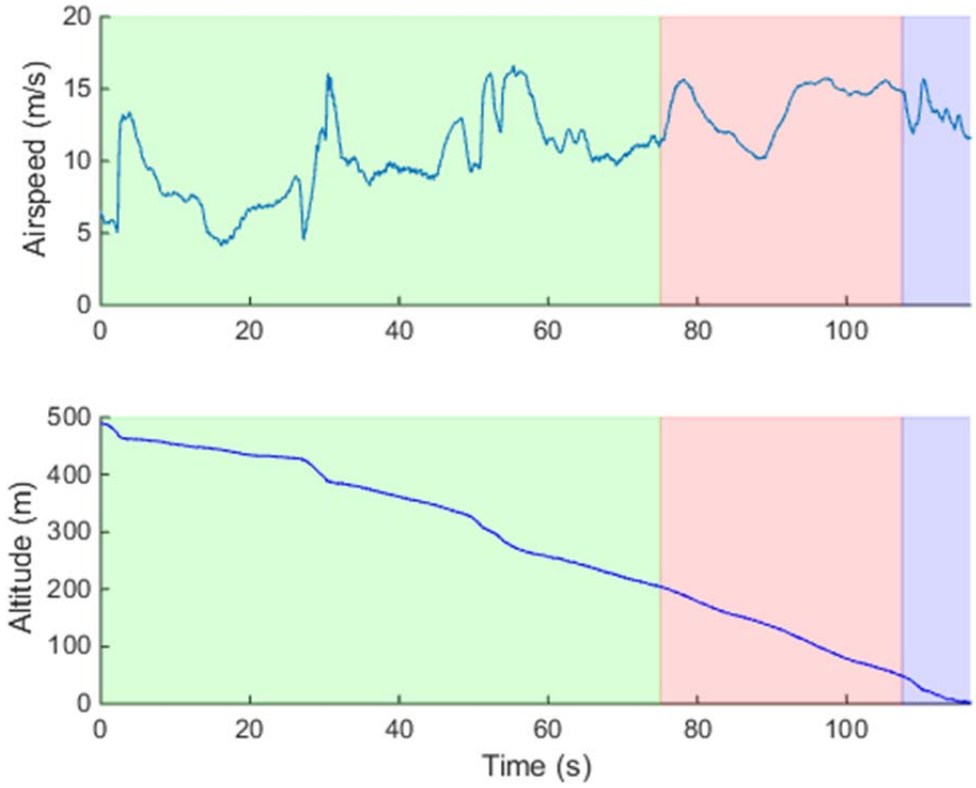

Figure 50. Sparrow Tuning Flight: Airspeed and Altitude Time Histories

The lack of control over airspeed can also be seen in Figure 50. While it is evident that the airspeed varies greatly during the flight, there is no data to support the conclusion that a lack of airspeed specifically causes the transition into uncontrollable flight as the first period begins at an airspeed of five meters per second and the second period begins at an airspeed of 10 meters per second while there are periods of controllable flight that have airspeed velocities less than either of those metrics.

The second flight in the data log, hereafter referred to as the "final flight," was also analyzed as shown below. Figures 51 and 52 illustrate a birds eye view and three-dimensional view of the flight path, respectively, and also show that the Sparrow was in RTL mode for the duration of the flight. After the tuning flight, it was evident that the PID gain values for roll, pitch and yaw were sufficient to continue into a more advanced analysis of orbit radius hold. The RTL flight mode automatically orders a spiral orbit pattern around the "home" point. Utilizing this pre-existing feature of APM Mission

Planner and MAVProxy, the ability of the Sparrow to execute the orbit flight and landing algorithm described in Chapter III of this thesis could be determined.

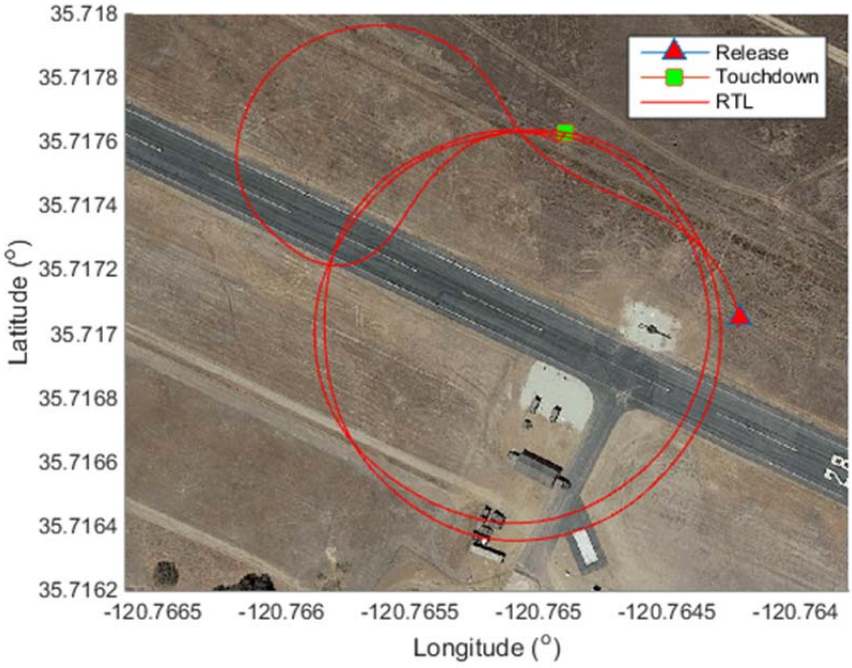

Figure 51. Sparrow Final Flight: Birds'-Eye View of Descent

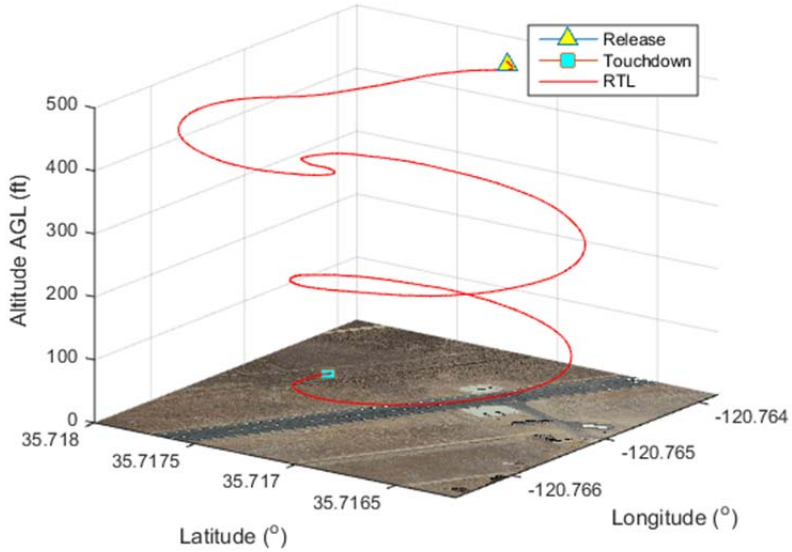

Figure 52. Sparrow Final Flight: Three-Dimensional View of Descent

Figure 53 shows the roll, pitch, and yaw rates for the final Sparrow flight. The oscillations in roll are explained by variations in the wind velocity, as seen in other flights; also, to hold an orbit, a bank angle must be applied to the aircraft. This bank angle is shown in Figure 54. While in the orbit phase of the flight, a right bank angle of approximately 10 to 30 degrees is maintained. However, the Sparrow was very stable in pitch and yaw. Figure 54 also shows the difference between the actual roll and the roll desired by the PIXHAWK AGU. While there are large oscillations in those values, they oscillate around zero and can likely be attributed to servo lag.

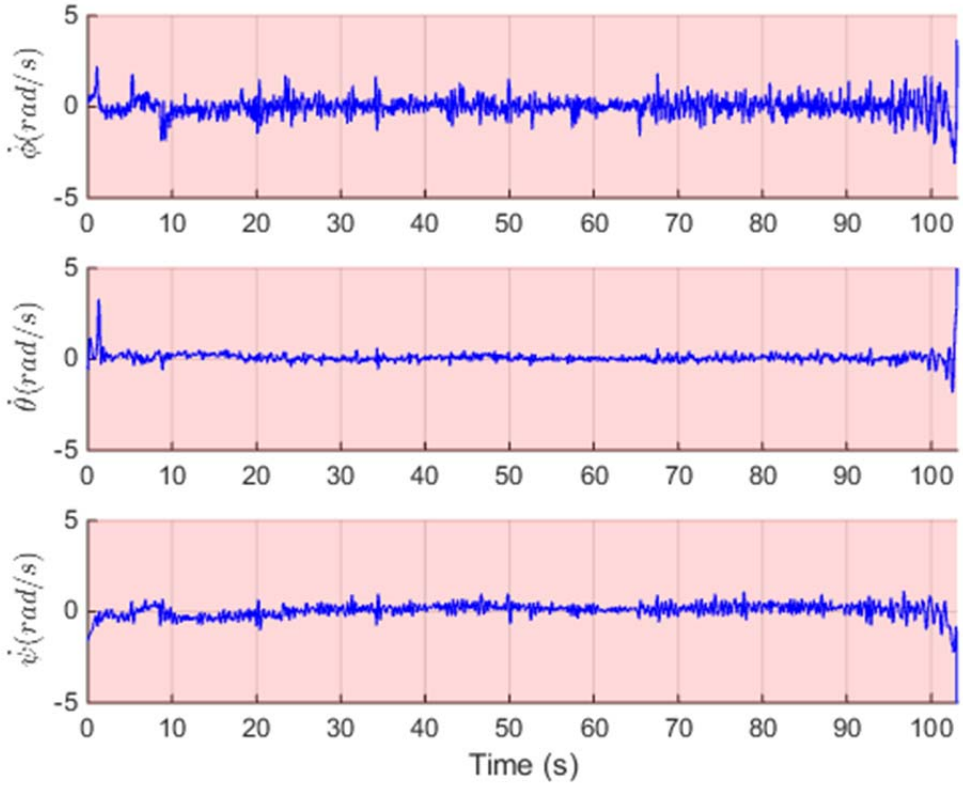

Figure 53. Sparrow Final Flight: Roll, Pitch and Yaw Rate Time Histories

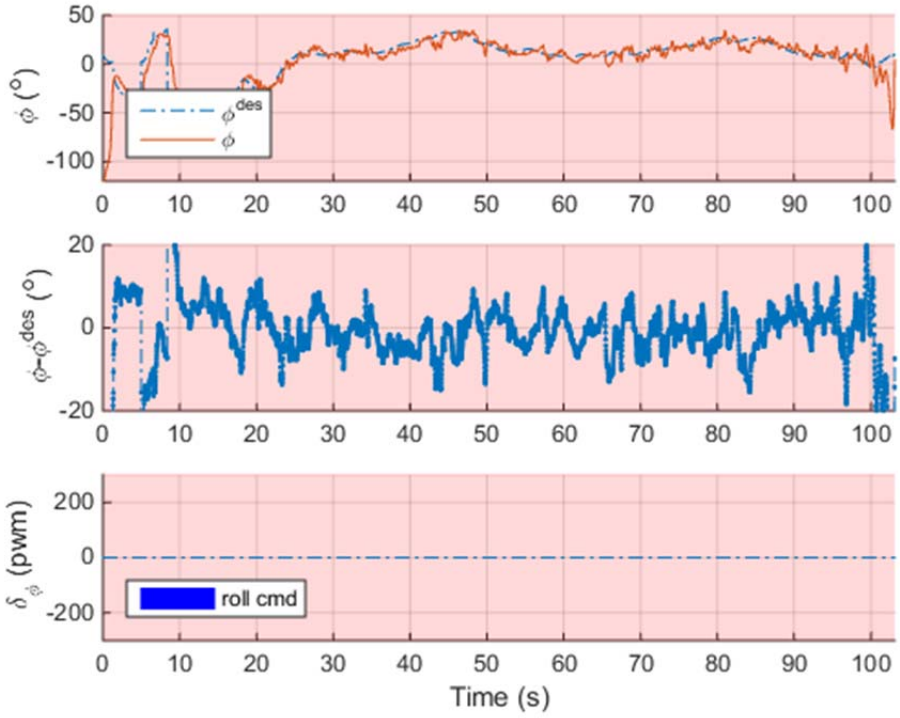

Figure 54. Sparrow Final Flight: Roll, Desired Roll and PWM Time Histories

Figure 55 shows the pitch, the difference between the pitch and the desired pitch and the PWM values. The measured pitch angle closely follows the desired pitch angle with a maximum error of ±5 degrees while in the orbit as seen in the middle plot. In order to solve the stalling issue that caused periods of uncontrollable flight discovered during the tuning flight, a minimum 10-degree down angle was commanded by changing the STAB_PITCH_DOWN parameter. The effect of this change can be seen in the uppermost plot. When comparing Figures 56 and 50, it is clear that there is a marked increase in average airspeed in the final flight. While the airspeed is not the causal link between normal and uncontrollable flight, it does play a significant role. Figure 57 shows the airspeed acceleration rates and descent velocities of the final flight. It is evident when comparing this plot to Figure 49 that the increased average airspeed also decreased the average change in airspeed and also removed major spikes in that average.

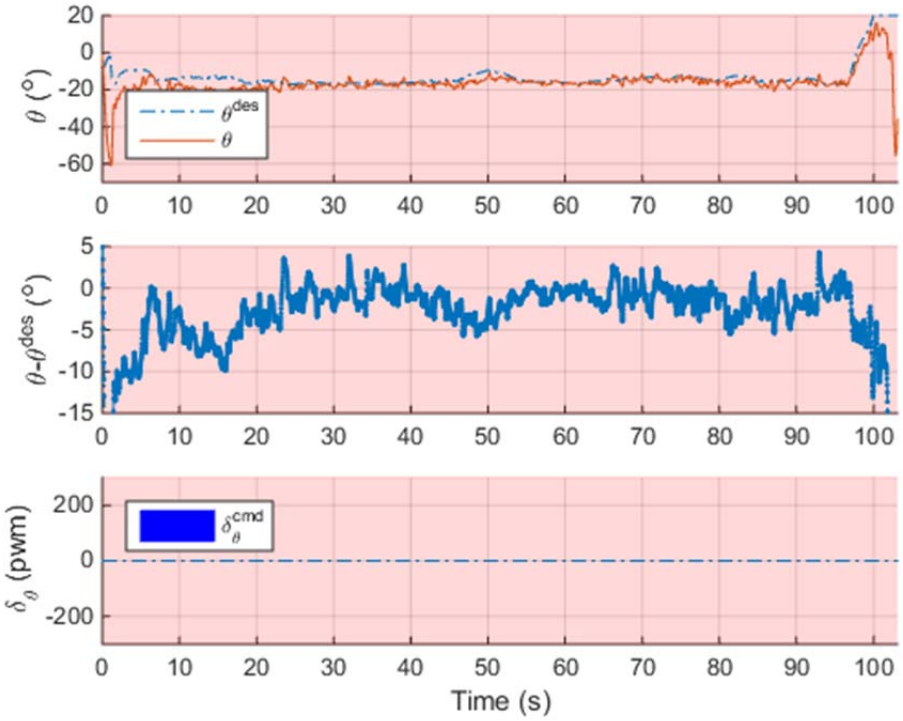

Figure 55. Sparrow Final Flight: Pitch, Desired Pitch and PWM Time Histories

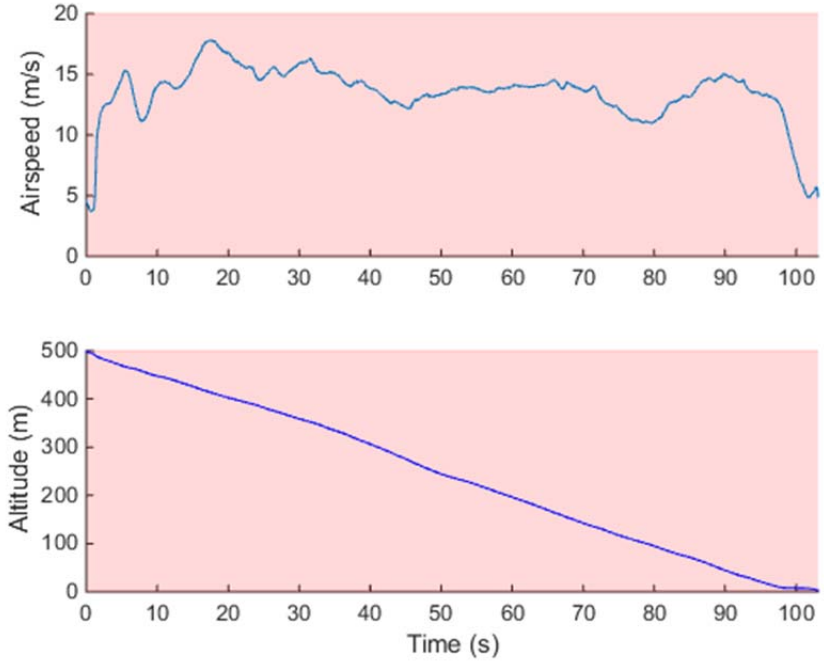

Figure 56. Sparrow Final Flight: Airspeed and Altitude Time Histories

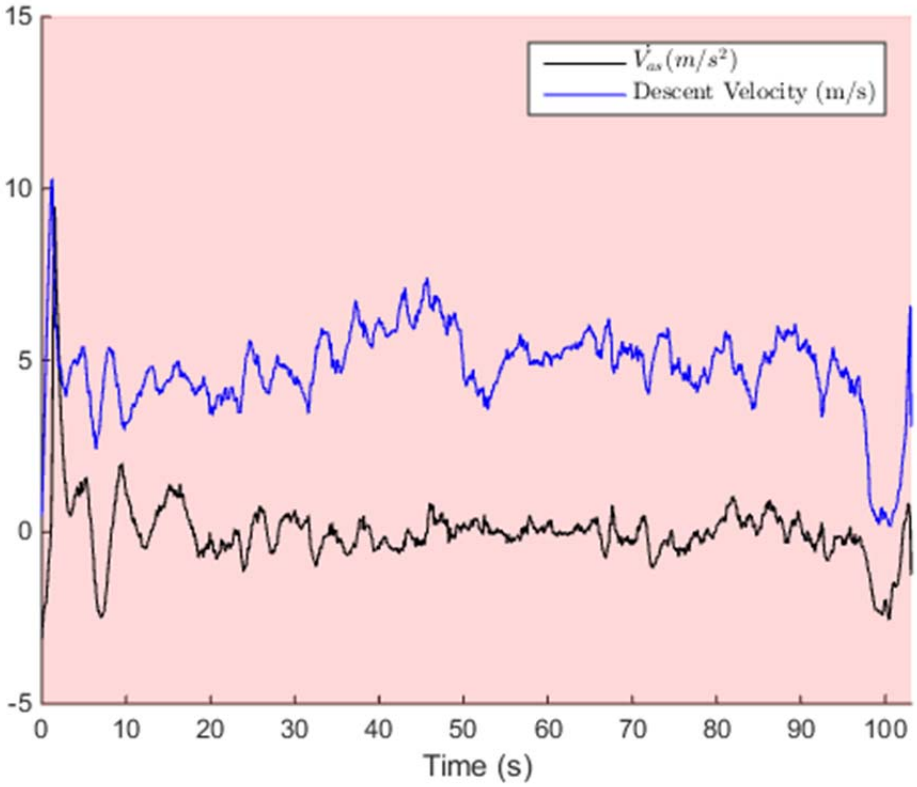

Figure 57. Sparrow Final Flight: Airspeed Acceleration Rate and Descent Velocity Correlation Time Histories

While the aforementioned changes to the parameters of the Sparrow positively impacted the stability of its flight characteristics, it did not significantly increase the glide slope. Figure 58 shows the sink rate and glide slope ratio data. While the sink rate did increase from 800 to 940 feet per minute, the lack of periods of uncontrollable flight resulted in an increase in glide slope from 2.1 to 2.9. This value, however, still does not meet the threshold values required by the MCWL as discussed in Chapter 2 of this thesis.

Finally, Figure 59 shows the ability of the Sparrow to hold a commanded orbit radius. While in the orbit, the error between the commanded radius and the actual flight path of the Sparrow was approximately nine meters on average. This is not sufficient accuracy to execute the orbit flight and landing maneuver described in Chapter III of this thesis.

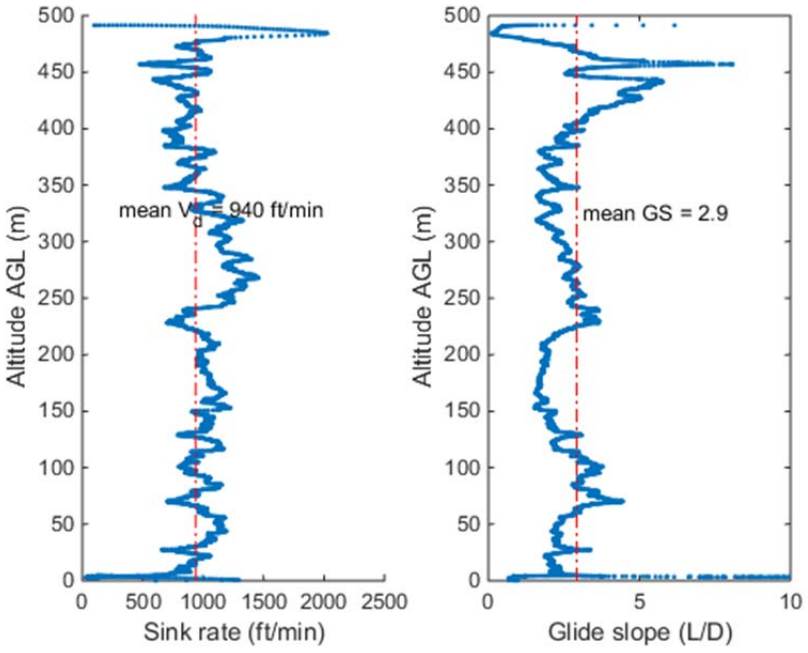

Figure 58. Sparrow Final Flight: Sink Rate and Glide Slope Versus Altitude

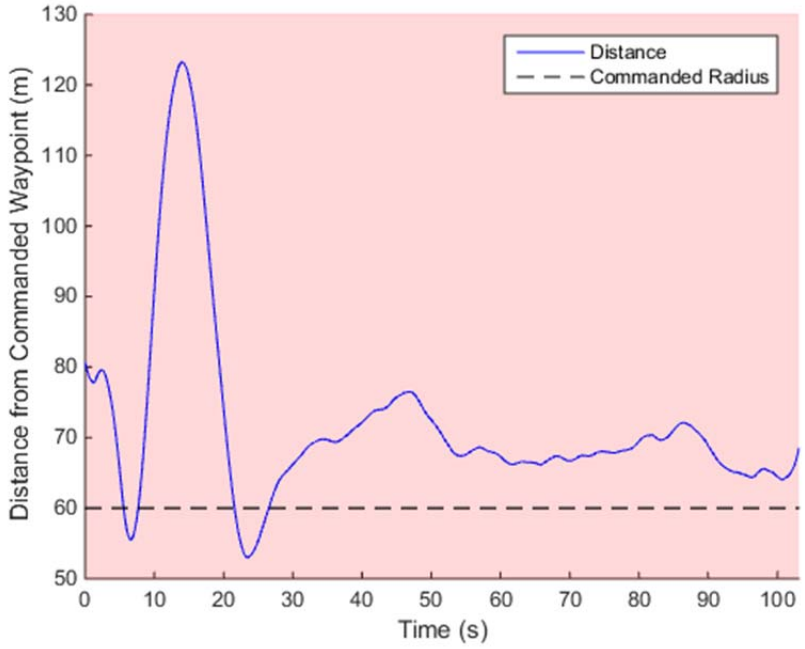

Figure 59. Sparrow Final Flight: Distance from Commanded Waypoint

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

Based on analysis and assessment of the flight test results, the author's overall conclusion is that a low-cost, glider-based PAD system utilizing COTS electronic components as described by the MCWL is likely a viable alternative to the parachute-based systems currently in use. In addition, the COTS components are likely to provide sufficient accuracy, reliability and durability to close the capability gap and meet the operational need for rapid-response, tactical logistical resupply in austere and dispersed locations. Also, modern manufacturing techniques are robust enough to create low-cost, fully functional gliding airframes that are both durable and reliable.

Logistic resupply gliders have an inherent requirement to be small and inexpensive to alleviate the warfighter from the burden of repacking and transporting previously used systems back to a centralized facility for refurbishment and reuse. It was determined that while the cost of a Gliding ADS is likely to meet the threshold requirement of $6 per pound of payload set by the MCWL at a payload weight of 500 pounds, it is highly unlikely that it will meet that requirement at lesser payload weights due to increased airframe costs. However, if the cost or complexity associated with COTS electronic systems or the materials and techniques used to manufacture airframes increases to the point where it can no longer be considered disposable, more research will be required to determine the economic viability of the proposed TACAD Supply Glider system.

Finally, research was not conducted on the effect of a loss of GPS signal on the COTS electronic components and the accompanying software suite or the accuracy of the internal gyros and accelerometers. While the system achieved a relatively high level of accuracy with all data streams available, any operationally deployed capability will be required to operate successfully in an environment where GPS or other data is not available. The inability to be flexible to available data streams will render a system useless to deployed units.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. MATLAB SCRIPTS

A. COST ANALYSIS SCRIPT

The following script was used to generate the plot in the cost analysis portion of this thesis.

```
%% Cost Estimation Script
% LT Chaz R. Henderson, USN
%%
close all, clear all, clc
rho_water = 62.30; % lb/ft^3 at room temperature (70 deg F)
cost_foam = 1/((20/12)*(30/12));    % dollar/sqft
cost_plywood = 0.42;                % dollar/sqft
cost_steel = 4.11;                  % dollar/sqft
cost_al = 14.57;                    % dollar/sqft
    FuselageH = 2.25/12;            % feet
    FuselageW = 2.0/12;             % feet
    FuselageL = 14.5/12;            % feet
    FuselageL1 = 6.0/12;            % feet
    WingL = 14.25*2/12;             % feet
    WingW = 6.25/12;                % feet
FuselageVolume = (FuselageH*FuselageW*FuselageL)-...
                 (0.4*FuselageL1*FuselageH*FuselageW);  % ft^3
FuselageWeight = FuselageVolume*rho_water;              % lb
WingArea = WingL*WingW;                                 % ft^2
WingLoad = FuselageWeight/WingArea;                     % lb/ft^2
    Weight = [1:500]';
for i=1:150
ServoCost(i)= 5.85*3;             % dollars
end
    for i=151:500
    ServoCost(i) = 30*3;          % dollars
    end
for i=1:length(Weight)
PIXHAWKcost(i) = 200;             % dollars
GPScost(i) = 90;                  % dollars
TRXcost(i) = 45;                  % dollars
cost_ALplane(i) = (Weight(i)/WingLoad)*2*cost_al;
cost_foamplane(i) = (Weight(i)/WingLoad)*2*cost_foam;
cost_steelplane(i) = (Weight(i)/WingLoad)*2*cost_steel;
cost_plywoodplane(i) = (Weight(i)/WingLoad)*2*cost_plywood;
end
    costF = PIXHAWKcost+GPScost+ServoCost+TRXcost+cost_foamplane;
    costP = PIXHAWKcost+GPScost+ServoCost+TRXcost+cost_plywoodplane;
    costS = PIXHAWKcost+GPScost+ServoCost+TRXcost+cost_steelplane;
%%
subplot(211)
area(Weight,[PIXHAWKcost',GPScost',ServoCost',TRXcost',cost_steelplane'
])...

%'FaceAlpha',0.5);
```

```
grid; hold on
plot(Weight,costP,'-.')
ylim([0 3000])
xlabel('Weight (lb)'), ylabel('Cost ($)')
legend('COTS autopilot','Sensors','Servos','Tx/Rx
Antenna','Airframe','Plywood Airframe',...

'location','northwest');
for i=1:length(Weight)
    costperweightF(i) = costF(i)/Weight(i);
    costperweightP(i) = costP(i)/Weight(i);
    costperweightS(i) = costS(i)/Weight(i);
end
%%
    subplot(212)

area(Weight,[PIXHAWKcost'./Weight,GPScost'./Weight,ServoCost'./Weight,.
..

TRXcost'./Weight,cost_steelplane'./Weight]),%'FaceAlpha',0.5);
    grid; hold on
    plot([0 500],6*[1 1],'r-.')
    ylabel('Cost/Weight (USD/lb)'), xlabel('Weight (lb)')
    legend('COTS autopilot','Sensors','Servos','Tx/Rx
Antenna','Airframe','Requirement',...

'location','northeast');
    ylim([0 30])
```

B. PUN-JET GLIDING ADS PROTOTYPE DATA PROCESSING AND ANALYSIS SCRIPT

The following script was used to process the first Pun-Jet Gliding ADS Prototype flights. It includes data parsing for Simulink PID simulations that were not conducted for the Sparrow Gliding ADS Prototype.

```
%Process Punjet data
close all, clear all, clc
CampRoberts = 1;
% Read in the data file you want to process that inclue YMDHM sequence:
%Note: First use the Mission Planner software to convert the log file
to;
%      a .mat file;
[filename, pathname] = uigetfile('*.mat','Choose PX4 data file');
FileName = [pathname filename];
iD=strfind(filename,'201');
YY=str2num(filename(iD:iD+3));     Mo=str2num(filename(iD+4:iD+5));
DD=str2num(filename(iD+6:iD+7));   HH=str2num(filename(iD+8:iD+9))-0*7;
Mi=str2num(filename(iD+10:iD+11));
DateNumber0=datenum(YY,Mo,DD,HH,Mi,0);
```

```
load(FileName);

%

acceleration = sqrt(IMU2(:,6).^2+IMU2(:,7).^2+IMU2(:,8).^2);
[Index_drop,~]=find(RCOU(:,9)<1300 & BARO(:,3)>100);
Index_drop = min(Index_drop);
a_total = interp1(IMU2(:,2),acceleration,RCOU(:,2),'pchip');
[Index_land,~]=find(a_total>20);
Index_land = max(Index_land);
time_of_release = RCOU(Index_drop(1),2);
time_of_land = RCOU(max(Index_land),2);

flight_time = (time_of_land - time_of_release )/(1*10^6)/60;


% Synchronize data with IMU2

index_start = find(IMU2(:,2)>=time_of_release,1);
index_end = find(IMU2(:,2)>=time_of_land,1);
time = (IMU2(index_start:index_end,2)-IMU2(index_start,2))/(1*10^6);

time_clean = [0:1/50:(length(time)-1)/50]';

p = IMU2(index_start:index_end,3); %roll rate
p = interp1(time,p,time_clean,'pchip');
q = IMU2(index_start:index_end,4); %pitch rate
q = interp1(time,q,time_clean,'pchip');
r = IMU2(index_start:index_end,5); %yaw rate
r = interp1(time,q,time_clean,'pchip');

BAlt=interp1(BARO(:,2),BARO(:,3),IMU2(index_start:index_end,2),'pchip')
;
des_Roll=interp1(ATT(:,2),ATT(:,3),IMU2(index_start:index_end,2),'pchip
');
Roll=interp1(ATT(:,2),ATT(:,4),IMU2(index_start:index_end,2),'pchip');
des_Pitch=interp1(ATT(:,2),ATT(:,5),IMU2(index_start:index_end,2),'pchi
p');
Pitch=interp1(ATT(:,2),ATT(:,6),IMU2(index_start:index_end,2),'pchip');
target_bearing=interp1(NTUN(:,2),NTUN(:,5),IMU2(index_start:index_end,2
),'pchip');
bearing=interp1(ATT(:,2),ATT(:,8),IMU2(index_start:index_end,2),'pchip'
);

ch_1_subtrim = 1490;
ch_2_subtrim = 1350;
ch_1 =
interp1(RCOU(:,2),RCOU(:,3),IMU2(index_start:index_end,2),'pchip')-
ch_1_subtrim;
ch_2 =
interp1(RCOU(:,2),RCOU(:,4),IMU2(index_start:index_end,2),'pchip')-
ch_2_subtrim;
```

```
roll_cmd = -(ch_1+ch_2)/2;
pitch_cmd = -(ch_1-ch_2)/2;

%  Slice data for roll model

IMU_dt = 1/50; %50 Hz
start_time = 207.6/IMU_dt;
end_time = 209.4/IMU_dt;

time = time_clean(start_time:end_time);
p=p(start_time:end_time);
roll_cmd=roll_cmd(start_time:end_time);

MODE(:,2) = MODE(:,2) - IMU2(index_start,2);

figure
subplot(2,1,1)
plot(time,roll_cmd)
hold on
for i=1:length(MODE)
    if MODE(i,4)==0;
        if (MODE(i,2)*10^-6) >= time(1) && (MODE(i,2)*10^-6) <=
time(end)
        line((MODE(i,2)*10^-6)*[1
1],[min(roll_cmd),max(roll_cmd)],'linestyle','--','color','r')
        end
    end
end
hold off
ylabel('\delta PWM'), xlabel('time (s)')
subplot(2,1,2)
plot(time,p)
ylabel('roll rate deg/s'), xlabel('time (s)')

save('roll_model_data.mat','time','p','roll_cmd')

% Plots
figure
subplot(3,1,1)
plot(time_clean,p)
subplot(3,1,2)
plot(time_clean,q)
subplot(3,1,3)
plot(time_clean,r)

for i=1:length(target_bearing)
    target_bearing(i) = wrapTo360(target_bearing(i)-90);
end

figure
subplot(2,1,1)
plot(time,des_Roll,time,Roll)
line([time(1),time(end)],[0,0],'linestyle','--','color','r')
xlabel('seconds')
```

```
ylabel('degrees')
legend('desired \phi','\phi')
%xlim([20,115])
subplot(2,1,2)
plot(time,roll_cmd)
line([time(1),time(end)],[0,0],'linestyle','--','color','r')
xlabel('seconds')
ylabel('\delta pwm')
legend('roll cmd')
%xlim([20,115])

figure
subplot(2,1,1)
plot(time,des_Pitch,time,Pitch)
line([time(1),time(end)],[0,0],'linestyle','--','color','r')
xlabel('seconds')
ylabel('degrees')
legend('desired \theta','\theta')
subplot(2,1,2)
plot(time,pitch_cmd)
line([time(1),time(end)],[0,0],'linestyle','--','color','r')
xlabel('seconds')
ylabel('pwm')
legend('pitch cmd')

% Bird-eye view

figure

CRImage = imread('CPRobertsflip','jpg');
DZimage = CRImage(:,:,1:3);
image([-120.788473 -120.758492],[35.714764 35.731265], DZimage)
hold on

latitude = AHR2(:,7);
longitude = AHR2(:,8);
altitude = AHR2(:,6);

plot(longitude,latitude,'.') %plot lat long
xlabel('Latitude, ^o'), ylabel('Longitude, ^o')
axis([-120.788473 -120.758492 35.714764 35.731265])
set(gca,'YDir','normal')
daspect([1 cosd(AHR2(end,7)) 0.5])
view(30,30)
hold off

figure
plot3(longitude,latitude,altitude,'.')
xlabel('longitude')
ylabel('latitude')
zlabel('altitude (ft msl)')
grid on

%  Slice data for pitch model
```

```
time_clean = [0:0.1:time(end)];
q_clean = interp1(time,q,time_clean,'pchip'); %pitch rate
pitch_cmd_clean = interp1(time,pitch_cmd,time_clean,'pchip'); %pitch
command

start_time = 510;
end_time = 530;%length(time);

time = time_clean(start_time:end_time);
q=q_clean(start_time:end_time);
pitch_cmd=pitch_cmd_clean(start_time:end_time);

save('pitch_model_data.mat','time','q','pitch_cmd')

% Density and descent rate

BPress=interp1(BARO(:,2),BARO(:,4),RCOU(Index_drop(1):Index_land(1),2),
'pchip');
BTemp=interp1(BARO(:,2),BARO(:,5),RCOU(Index_drop(1):Index_land(1),2),'
pchip');
dens=BPress./287./(BTemp+273.15);
VD=(interp1(EKF1(:,2),EKF1(:,8),RCOU(Index_drop(1):Index_land(1),2),'pc
hip'))*(3.2808399*60);
VN=(interp1(EKF1(:,2),EKF1(:,6),RCOU(Index_drop(1):Index_land(1),2),'pc
hip'))*(3.2808399*60);
VE=(interp1(EKF1(:,2),EKF1(:,7),RCOU(Index_drop(1):Index_land(1),2),'pc
hip'))*(3.2808399*60);

VNE=sqrt(VN.^2+VE.^2);
GSlope=VNE./VD;

figure
hold on
plot(dens,altitude,'.') %density vs altitude
densISA=1.225*(288.15./(288.15-6.5*(altitude/3.2808399/1000))).^(-
34.163195/6.5);
plot(densISA,altitude,'-.')
xlabel('Computed \rho, kg/m^3'), ylabel('Altitude MSL, ft')
h=legend('Computed','ISA');
set(h,'fontsize',8);
ylabel('Altitude MSL, ft'), xlabel('\rho, kg/m^3')
xlim([0.9 1.3])
hold off

subplot(121)
hold on
mVD=mean(VD);
plot(VD,altitude,'.')
Y=axis;
plot(mVD*[1 1],Y(3:4),'-.r')
xlabel('Sink rate, ft/min'), ylabel('Altitude MSL, ft')
text(mVD-1,0.85*Y(4)+0.03*Y(3),['mean V_d = ' num2str(mVD,3) '
ft/min'])
```

```
ylim([900 2000])
hold off

subplot(122)
plot(GSlope(20:end),altitude(20:end),'.')
line([mean(GSlope(20:end))*[1,1]],[0 2000],'linestyle','-
.','color','r')
text(1.9,1800,['mean glide slope = ' num2str(mean(GSlope(20:end)),3)
''])
ylabel('Altitude MSL, ft'), xlabel('glide slope')
xlim([0 6])
ylim([900 2000])
```

C. WRAP TO 360 FUNCTION

The following function is used in the Pun-Jet Gliding ADS Prototype Data Processing and Analysis script. It transforms angular values that fall outside of the zero to 360 degree normal.

```
function [ angle_out ] = wrapTo360( angle )
%WRAPTO360 Summary of this function goes here
%   Detailed explanation goes here

if (angle > 360)
    angle = angle-360;
end

if (angle < 0)
    angle = angle +360;
end

angle_out = angle;
end
```

D. SPARROW GLIDING ADS PROTOTYPE ANALYSIS SCRIPT

The following script was used to analyze the Sparrow Gliding ADS Prototype experimental flight logs. While it is similar in many respects to the Pun-Jet Gliding ADS Prototype analysis script, it features advanced data parsing procedures and additional analysis tools not featured in previous versions.

```
%  Sparrow Gliding ADS Prototype Analysis Script
%  Chaz R. Henderson, LT, USN

%  This script was adapted from previous work by LT Ryan Beall, USN, and
```

```
%  Dr. Oleg Yakimenko of the Naval Postgraduate School Systems
Engineering
%  (SE) Department. It was modified from the Pun-Jet Gliding ADS
%  analysis script because the Pixhawk logged more than a single flight
on
%  each data log.

clear
clc
close all

% Import data file for analysis

[filename, pathname] = uigetfile('*.m','Choose first MATLAB file');
run(filename)

% Initiate variables and interpolate required data

testsite = 'McMillan Airfield, Camp Roberts, CA';
Date = 'Thursday, October 20th, 2016';
FlightModes = {'Release','Touchdown','Manual','FBWA','RTL'};
ch_2_subtrim = 1392;
ch_4_subtrim = 1522;

BAlt = interp1(BARO.data(:,2),BARO.data(:,3),IMU2.data(:,2),'pchip');
des_Roll = interp1(ATT.data(:,2),ATT.data(:,3),IMU2.data(:,2),'pchip');
Roll = interp1(ATT.data(:,2),ATT.data(:,4),IMU2.data(:,2),'pchip');
des_Pitch = interp1(ATT.data(:,2),ATT.data(:,5),IMU2.data(:,2),'pchip');
Pitch = interp1(ATT.data(:,2),ATT.data(:,6),IMU2.data(:,2),'pchip');
target_bearing =
interp1(NTUN.data(:,2),NTUN.data(:,5),IMU2.data(:,2),'pchip');
bearing = interp1(ATT.data(:,2),ATT.data(:,8),IMU2.data(:,2),'pchip');
accel = sqrt(IMU2.data(:,6).^2+IMU2.data(:,7).^2+IMU2.data(:,8).^2);
a_total = interp1(IMU2.data(:,2),accel,IMU2.data(:,2),'pchip');
latitude = interp1(GPS.data(:,2),GPS.data(:,8),IMU2.data(:,2),'pchip');
longitude = interp1(GPS.data(:,2),GPS.data(:,9),IMU2.data(:,2),'pchip');
altitude = interp1(GPS.data(:,2),GPS.data(:,10),IMU2.data(:,2),'pchip');
BPress = interp1(BARO.data(:,2),BARO.data(:,4),IMU2.data(:,2),'pchip');
BTemp = interp1(BARO.data(:,2),BARO.data(:,5),IMU2.data(:,2),'pchip');
VD =
(interp1(NKF1.data(:,2),NKF1.data(:,8),IMU2.data(:,2),'pchip'))*(3.28083
99*60);
VD1 = interp1(NKF1.data(:,2),NKF1.data(:,8),IMU2.data(:,2),'pchip');
VN =
(interp1(NKF1.data(:,2),NKF1.data(:,6),IMU2.data(:,2),'pchip'))*(3.28083
99*60);
VN1 = interp1(NKF1.data(:,2),NKF1.data(:,6),IMU2.data(:,2),'pchip');
VE =
(interp1(NKF1.data(:,2),NKF1.data(:,7),IMU2.data(:,2),'pchip'))*(3.28083
99*60);
VE1 = interp1(NKF1.data(:,2),NKF1.data(:,7),IMU2.data(:,2),'pchip');
VWN = interp1(NKF2.data(:,2),NKF2.data(:,7),IMU2.data(:,2),'pchip');
VWE = interp1(NKF2.data(:,2),NKF2.data(:,8),IMU2.data(:,2),'pchip');

Vas = sqrt((VN1-VWN).^2 + (VE1-VWE).^2);
wpdist = interp1(NTUN.data(:,2),NTUN.data(:,3),IMU2.data(:,2),'pchip');
airspeed =
interp1(TECS.data(:,2),TECS.data(:,8),IMU2.data(:,2),'pchip');
Time = (IMU2.data(:,2)-IMU2.data(1,2))/(1*10^6);

% Clean flight mode (channel 6) and release mechanism servo (channel 7)
```

```
% PWM input data

r = 1;
while r<length(RCIN.data(:,9))
    if RCIN.data(r,9) == 0
        RCIN.data(r,9) = NaN;
    end
    if RCIN.data(r,8) == 0
        RCIN.data(r,8) = NaN;
    end
    r = r+1;
end

% Interpolate flight mode (channel 6) and release mechanism servo
(channel
% 7) PWM input data

CH6 = interp1(RCIN.data(:,2),RCIN.data(:,8),IMU2.data(:,2),'pchip');
CH7 = interp1(RCIN.data(:,2),RCIN.data(:,9),IMU2.data(:,2),'pchip');

% Plot used to tune triggers and determine which flight will be used for
% analysis

figure(1)
plot(IMU2.data(:,2),BAlt,IMU2.data(:,2),CH7,IMU2.data(:,2),(a_total-
9.81)*7)
title({'Sparrow Snowflake G ADS Log Overview';testsite;Date});
legend('Barometric Alt','Release PWM','Acceleration');

% Determine the initial drop point for each flight by finding the
decrease in
% PWM in channel 7 (release mechanism) when the altitude is above 300m

[I_drop,~] = find(RCIN.data(:,9)<1250 & BARO.data(:,3)>300);
[I_d,~] = find(diff(I_drop)>100);
I_d = I_d+1;
t_drop(1) = I_drop(1);
n = 1;
while n <= length(I_d)
    t_drop(n+1) = I_drop(I_d(n));
    n = n+1;
end
t_drop = t_drop.';

% Determine the landing point by determining when the airframe altitude
% drops below 1.5 meters AGL using barometric data

[I_land,~]=find(BARO.data(:,3)<3);
[I_l,~] = find(diff(I_land)>500);
I_l = I_l+1;
k = 1;
while k <= length(I_l)
    t_land(k) = I_land(I_l(k));
    k = k+1;
end
t_land = t_land.';

% Make array of times of drop and landing for each flight

t_flight = cat(2,t_drop,t_land);
```

```
% Ask user which flight is to be analyzed

fprintf('Which flight would you like to analyze? [Pick a number between
1 and %i]\n',n);
i_flight = input('');

% Determine which index points in the arrays are going to be plotted for
% analysis

time_of_release = RCIN.data(t_flight(i_flight,1),2);
time_of_land = RCIN.data(t_flight(i_flight,2),2);

flight_time = (time_of_land - time_of_release )/(1*10^6)/60;

index_start = find(IMU2.data(:,2)>=time_of_release,1);
index_end = find(IMU2.data(:,2)>=time_of_land,1);
time = (IMU2.data(index_start:index_end,2)-
IMU2.data(index_start,2))/(1*10^6);

time_clean = [0:1/50:(length(time)-1)/50]';
Time = Time-Time(index_start);

% Determine the flight modes executed and times entered for the selected
% flight

FlightMode = unique(MODE.data(:,3));
c = 1;
while c<=length(MODE.data(:,2))
    [~, I_FM(c)] = min(abs(IMU2.data(:,2)-MODE.data(c,2)));
    c = c+1;
end
I_FM = unique(I_FM);
I_FM = I_FM.';

P_FM = ones(1,length(I_FM));
d = 1;
while d<=length(I_FM)
    P_FM(d) = CH6(I_FM(d));
    d = d+1;
end
P_FM = P_FM.';
mode_data = cat(2,I_FM,P_FM);
mode_data(length(I_FM)+1,1) = length(IMU2.data(:,2));
mode_data(length(I_FM)+1,2) = mode_data(length(I_FM),2);

% Birds-eye view of the descent with release and landing points. The
line
% color will change based on the mode associated with that portion of
the
% flight

figure(2)
E = {'h1,','h2,','h3,','h4,','h5,'};
hold on
h1 =
plot(longitude(index_start),latitude(index_start),'Marker','^','MarkerFa
ceColor','red','MarkerSize',9);
F(1) = 1;
h2 =
plot(longitude(index_end),latitude(index_end),'Marker','s','MarkerFaceCo
lor','green','MarkerSize',9);
```

```
F(2) = 1;

for  g=index_start:index_end-1
    if CH6(g) == 1099
        h3 = plot(longitude(g:g+1),latitude(g:g+1),'-','Color','blue');
        F(3) = 1;
    elseif CH6(g) == 1500
        h4 = plot(longitude(g:g+1),latitude(g:g+1),'-','Color','green');
        F(4) = 1;
    elseif CH6(g) == 1901
        h5 = plot(longitude(g:g+1),latitude(g:g+1),'-','Color','red');
        F(5) = 1;
    end
end

ind=find(F);
eval(['u=[' [E{ind}] ']'])
legend(u,FlightModes(ind))
plot_google_map('MapType','satellite');
[aa bb cc]=plot_google_map('MapType','satellite');
%title({'Sparrow Snowflake G ADS Birds Eye View';testsite;Date});
ylabel('Latitude (^o)')
xlabel('Longitude (^o)')
YY=axis;
hold off

% 3-D plot of the descent with release and landing points. The color of
the
% line will change with respect to the mode associated with that portion
% of the flight

figure(3)
Alt=altitude(index_start:index_end)-altitude(index_end);
L = {'h6,','h7,','h8,','h9,','h10,'};
hold on
h6 =
plot3(longitude(index_start),latitude(index_start),Alt(1),'Marker','^','
MarkerFaceColor','yellow','MarkerSize',9);
flag(1) = 1;
h7 =
plot3(longitude(index_end),latitude(index_end),Alt(end),'Marker','s','Ma
rkerFaceColor','cyan','MarkerSize',9);
flag(2) = 1;

for  m=index_start:index_end-1
    if CH6(m) == 1099
        h8 = plot3(longitude(m:m+1),latitude(m:m+1),BAlt(m:m+1),'-
','Color','blue');
        flag(3) = 1;
    elseif CH6(m) == 1500
        h9 = plot3(longitude(m:m+1),latitude(m:m+1),BAlt(m:m+1),'-
','Color','green');
        flag(4) = 1;
    elseif CH6(m) == 1901
        h10 = plot3(longitude(m:m+1),latitude(m:m+1),BAlt(m:m+1),'-
','Color','red');
        flag(5) = 1;
    end
end

ind=find(flag);
eval(['j=[' [L{ind}] ']'])
```

```
legend(j,FlightModes(ind))

%title({'Sparrow Snowflake G ADS 3-D View of Descent';testsite;Date});
ylabel('Latitude (^o)')
xlabel('Longitude (^o)')
zlabel('Altitude AGL (ft)')
grid on
xlim(YY(1:2)); ylim(YY(3:4));
image(aa,bb,cc)
hold off

% Roll, Pitch and Yaw Rate plots

R_dot = IMU2.data(:,3);
P_dot = IMU2.data(:,4);
Y_dot = IMU2.data(:,5);

ch_2 = interp1(RCIN.data(:,2),RCIN.data(:,4),IMU2.data(:,2),'pchip')-
ch_2_subtrim;
ch_4 = interp1(RCIN.data(:,2),RCIN.data(:,6),IMU2.data(:,2),'pchip')-
ch_4_subtrim;

roll_cmd = -(ch_2+ch_4)/2;
pitch_cmd = -(ch_2-ch_4)/2;

figure(4)
subplot(3,1,1)
%title({'Sparrow Snowflake G ADS Roll, Pitch and Yaw
Rates';testsite;Date});
hold on
Q = {'','','MANarea,','FBWAarea,','RTLarea,'};
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),50*[-1 -1 1 1],'b');

set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
        flag1(3) = 1;
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),50*[-1 -1 1 1],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
        flag1(4) = 1;
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),50*[-1 -1 1 1],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
        flag1(5) = 1;
    end
end
plot(Time(index_start:index_end),R_dot(index_start:index_end),'blue')
ylabel('$\dot{\phi} (rad/s)$','Interpreter','latex')
grid
xlim(Time([index_start index_end]))
ylim(50*[-1 1])
hold off

subplot(3,1,2)
```

```
hold on
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),40*[-1 -1 1 1],'b');

set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),40*[-1 -1 1 1],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),40*[-1 -1 1 1],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
end
plot(Time(index_start:index_end),P_dot(index_start:index_end),'blue')
ylabel('$\dot{\theta} (rad/s)$','Interpreter','latex')
grid
xlim(Time([index_start index_end]))
ylim(40*[-1 1])
hold off

subplot(3,1,3)
hold on
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),40*[-1 -1 1 1],'b');

set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),40*[-1 -1 1 1],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),40*[-1 -1 1 1],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
end
plot(Time(index_start:index_end),Y_dot(index_start:index_end),'blue')
ylabel('$\dot{\psi} (rad/s)$','Interpreter','latex')
grid
xlim(Time([index_start index_end]))
ylim(40*[-1 1])
xlabel('Time (s)')
%index=find(flag1);
%eval(['z=[' [Q{index}] ']'])
%legend(z,FlightModes(index),'Location','bestoutside','Orientation','hor
izontal')
hold off

% Roll, Desired Roll and Roll Command plots
```

```
figure(5)
subplot(3,1,1)
hold on
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-120 -120 50 50],'b');

set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-120 -120 50 50],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-120 -120 50 50],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
end
plot(Time(index_start:index_end),des_Roll(index_start:index_end),'-
.',Time(index_start:index_end),Roll(index_start:index_end))
%line([time(1),time(end)],[0,0],'linestyle','--','color','g')
xlim(Time([index_start index_end]))
ylim([-120 50])
ylabel('\phi (^o)')
legend('\phi^{des}','\phi','location','s')
grid
hold off

subplot(3,1,2)
hold on
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),20*[-1 -1 1 1],'b');

set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),20*[-1 -1 1 1],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),20*[-1 -1 1 1],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
end
plot(Time(index_start:index_end),Roll(index_start:index_end)-
des_Roll(index_start:index_end),'.-.')
ylabel('\phi-\phi^{des} (^o)')
xlim(Time([index_start index_end]))
ylim(20*[-1 1])
grid
hold off
```

```
subplot(3,1,3)
hold on
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-300 -300 300 300],'b');

set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-300 -300 300 300],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-300 -300 300 300],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
end
plot(Time(index_start:index_end),roll_cmd(index_start:index_end),'-.')
%line([time(1),time(end)],[0,0],'linestyle','--','color','g')
xlabel('Time (s)')
ylabel('\delta_{\phi} (pwm)')
ylim([-300 300])
xlim(Time([index_start index_end]))
legend('roll cmd','location','s')
grid
hold off

% Pitch, Desired Pitch, and Pitch Command Plots

figure(6)
subplot(3,1,1)
hold on
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-70 -70 20 20],'b');

set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-70 -70 20 20],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-70 -70 20 20],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
end
plot(Time(index_start:index_end),des_Pitch(index_start:index_end),'-
.',Time(index_start:index_end),Pitch(index_start:index_end))
grid
%line([time(1),time(end)],[0,0],'linestyle','--','color','g')
```

```
ylabel('\theta (^o)')
legend('\theta^{des}','\theta','location','northwest')
ylim([-70 20])
xlim(Time([index_start index_end]))
hold off

subplot(3,1,2)
hold on
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-15 -15 5 5],'b');

set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-15 -15 5 5],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-15 -15 5 5],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
end
plot(Time(index_start:index_end),Pitch(index_start:index_end)-
des_Pitch(index_start:index_end),'.-.')
ylabel('\theta-\theta^{des} (^o)')
xlim(Time([index_start index_end]))
ylim([-15 5])
grid
hold off

subplot(3,1,3)
hold on
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-300 -300 300 300],'b');

set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-300 -300 300 300],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-300 -300 300 300],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
end
plot(Time(index_start:index_end),pitch_cmd(index_start:index_end),'-.')
grid
xlim(Time([index_start index_end]))
ylim([-300 300])
xlabel('Time (s)')
```

```
ylabel('\delta_{\theta} (pwm)')
legend('\delta_{\theta}^{cmd}','location','northwest')
hold off

% Density and descent rate

dens = BPress./287./(BTemp+273.15);
VNE=sqrt(VN.^2+VE.^2);
GSlope=VNE./VD;

figure(7)
subplot(1,2,1)
hold on
plot(dens(index_start:index_end),Alt,'.') %density vs altitude
densISA=1.225*(288.15./(288.15-6.5*(altitude/3.2808399/1000))).^(-
34.163195/6.5);
plot(densISA(index_start:index_end),Alt,'-.')
xlabel('Computed \rho (kg/m^3)'), ylabel('Altitude AGL (ft)')
h = legend('Computed','ISA','location','n');
set(h,'fontsize',8);
grid
hold off

% Glide Slope plots

figure(8)
subplot(1,2,1)
hold on
mVD=mean(VD(index_start:index_end));
plot(VD(index_start:index_end),Alt,'.')
Y=axis;
plot(mVD*[1 1],Y(3:4),'-.r')
xlabel('Sink rate (ft/min)'), ylabel('Altitude AGL (ft)')
text(300,0.65*Y(4)+0.35*Y(3),['mean V_d = ' num2str(mVD,'%.0f') '
ft/min'])
%ylim([0 1000])
xlim([0 2500])
hold off

subplot(1,2,2)
plot(GSlope(index_start:index_end),Alt,'.'), hold
Y=axis;
mGS=mean(GSlope(index_start:index_end));
plot(mGS*[1 1],Y(3:4),'-.r')
text(mGS+.2,0.65*Y(4)+0.35*Y(3),['mean GS = ' num2str(mGS,'%.1f') ''])
ylabel('Altitude AGL (ft)'), xlabel('Glide slope (L/D)')
xlim([0 10])
%ylim([0 1000])

% Determine the speed at which the airframe goes into uncontrolled
flight
% (stall) by comparing groundspeed to altitude

figure(9)
subplot(2,1,1)
hold on
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[0 0 20 20],'b');
```

```
set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[0 0 20 20],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[0 0 20 20],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
end
plot(Time(index_start:index_end),[Vas(index_start:index_end)])
ylabel('Airspeed (m/s)')
xlim(Time([index_start index_end]))
ylim([0 20])
hold off
%title({'Sparrow Snowflake G ADS Groundspeed v.
Altitude';testsite;Date});

subplot(2,1,2)
hold on
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[0 0 500 500],'b');

set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[0 0 500 500],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[0 0 500 500],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
end
plot(Time(index_start:index_end),BAlt(index_start:index_end),'blue')
ylabel('Altitude (m)')
xlabel('Time (s)')
xlim(Time([index_start index_end]))
ylim([0 500])
hold off

% Determine the distance from the commanded waypoint using WpDist data
from
% the NTUN data set

figure(10)
hold on
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[50 50 200 200],'b');
```

```
set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[50 50 200 200],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[50 50 200 200],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
end
h12 =
plot(Time(index_start:index_end),wpdist(index_start:index_end),'blue');
%title({'Sparrow Snowflake G ADS Orbit Radius Distance';testsite;Date});
ylabel('Distance from Commanded Waypoint (m)')
xlabel('Time (s)')
h13 =
line([0,length(wpdist(index_start:index_end))/50],[60,60],'linestyle','-
-','color','black');
legend([h12,h13],{'Distance','Commanded Radius'});
xlim(Time([index_start index_end]))
ylim([50 200])
hold off

% Rate of Airspeed Increase/Decrease (Acceleration) and Descent Velocity
% plots

% Take derivative of Airspeed
dt = 1/50;
for i=1:length(VD)-1
    Vas_dot(i) = (Vas(i+1)-Vas(i))/dt;
end

% Low Pass Filter

alpha = 0.95;
Vas_dot_filt(1) = Vas_dot(1);
for i=1:length(Vas_dot)
    Vas_dot_filt(i+1) = (Vas_dot(i)*(1-
alpha))+((alpha)*Vas_dot_filt(i));
end
Vas_dot_filt = Vas_dot_filt';

figure(11)
hold on
for q=1:length(mode_data)-1;
    if mode_data(q,2)==1099
        MANarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-5 -5 15 15],'b');

set(MANarea,'EdgeColor','blue','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
    if mode_data(q,2)==1500
        FBWAarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-5 -5 15 15],'g');

set(FBWAarea,'EdgeColor','green','EdgeAlpha',0.15,'FaceAlpha',0.15);
```

```
    end
    if mode_data(q,2)==1901
        RTLarea = patch(Time([mode_data(q,1) mode_data(q+1,1)-1
mode_data(q+1,1)-1 mode_data(q,1)]),[-5 -5 15 15],'r');

set(RTLarea,'EdgeColor','red','EdgeAlpha',0.15,'FaceAlpha',0.15);
    end
end
h10 =
plot(Time(index_start:index_end),Vas_dot_filt(index_start:index_end),'bl
ack');
h11 =
plot(Time(index_start:index_end),VD1(index_start:index_end),'blue');
xlim(Time([index_start index_end]))
ylim([-5 15])
xlabel('Time (s)')
legend([h10,h11],{'$\dot{V_{as}} (m/s^{2})$','Descent Velocity
(m/s)'},'Interpreter','latex')
```

E. PLOT GOOGLE MAP FUNCTION

The following function was used in the Snowflake data processing and analysis script. It was written by Zohar Bar-Yehuda and was last updated on July 13, 2016. It can be accessed at the following web address:

https://www.mathworks.com/matlabcentral/fileexchange/27627-zoharby-plot-google-map?requestedDomain=www.mathworks.com

```
function varargout = plot_google_map(varargin)
% function h = plot_google_map(varargin)
% Plots a google map on the current axes using the Google Static Maps
API
%
% USAGE:
% h = plot_google_map(Property, Value,...)
% Plots the map on the given axes. Used also if no output is specified
%
% Or:
% [lonVec latVec imag] = plot_google_map(Property, Value,...)
% Returns the map without plotting it
%
% PROPERTIES:
%    Axis           - Axis handle. If not given, gca is used.
%    Height (640)   - Height of the image in pixels (max 640)
%    Width  (640)   - Width of the image in pixels (max 640)
%    Scale (2)      - (1/2) Resolution scale factor. Using Scale=2 will
%                     double the resolution of the downloaded image (up
%                     to 1280x1280) and will result in finer rendering,
%                     but processing time will be longer.
%    Resize (1)     - (recommended 1-2) Resolution upsampling factor.
```

```
%                     Increases image resolution using imresize(). This
results
%                     in a finer image but it needs the image
processing
%                     toolbox and processing time will be longer.
%    MapType        - ('roadmap') Type of map to return. Any of
[roadmap,
%                     satellite, terrain, hybrid]. See the Google Maps
API for
%                     more information.
%    Alpha (1)      - (0-1) Transparency level of the map (0 is fully
%                     transparent). While the map is always moved to
the
%                     bottom of the plot (i.e. will not hide previously
%                     drawn items), this can be useful in order to
increase
%                     readability if many colors are plotted
%                     (using SCATTER for example).
%    ShowLabels (1) - (0/1) Controls whether to display city/street
textual labels on the map
%    Style          - (string) A style configuration string. See:
%
https://developers.google.com/maps/documentation/static-
maps/?csw=1#StyledMaps
%                     http://instrument.github.io/styled-maps-wizard/
%    Language       - (string) A 2 letter ISO 639-1 language code for
displaying labels in a
%                     local language instead of English (where
available).
%                     For example, for Chinese use:
%                     plot_google_map('language','zh')
%                     For the list of codes, see:
%                     http://en.wikipedia.org/wiki/List_of_ISO_639-
1_codes
%    Marker         - The marker argument is a text string with fields
%                     conforming to the Google Maps API. The
%                     following are valid examples:
%                     '43.0738740,-70.713993' (default midsize orange
marker)
%                     '43.0738740,-70.713993,blue' (midsize blue
marker)
%                     '43.0738740,-70.713993,yellowa' (midsize yellow
%                     marker with label "A")
%                     '43.0738740,-70.713993,tinyredb' (tiny red marker
%                     with label "B")
%    Refresh (1)    - (0/1) defines whether to automatically refresh
the
%                     map upon zoom/pan action on the figure.
%    AutoAxis (1)   - (0/1) defines whether to automatically adjust the
axis
%                     of the plot to avoid the map being stretched.
%                     This will adjust the span to be correct
%                     according to the shape of the map axes.
%    FigureResizeUpdate (1) - (0/1) defines whether to automatically
refresh the
```

```
%                     map upon resizing the figure. This will ensure
map
%                     isn't stretched after figure resize.
%    APIKey         - (string) set your own API key which you obtained
from Google:
%
http://developers.google.com/maps/documentation/staticmaps/#api_key
%                     This will enable up to 25,000 map requests per
day,
%                     compared to a few hundred requests without a key.
%                     To set the key, use:
%
plot_google_map('APIKey','SomeLongStringObtaindFromGoogle')
%                     You need to do this only once to set the key.
%                     To disable the use of a key, use:
%                     plot_google_map('APIKey','')
%
% OUTPUT:
%    h              - Handle to the plotted map
%
%    lonVect        - Vector of Longidute coordinates (WGS84) of the
image
%    latVect        - Vector of Latidute coordinates (WGS84) of the
image
%    imag           - Image matrix (height,width,3) of the map
%
% EXAMPLE - plot a map showing some capitals in Europe:
%    lat = [48.8708   51.5188   41.9260   40.4312   52.523   37.982];
%    lon = [2.4131    -0.1300    12.4951   -3.6788    13.415   23.715];
%    plot(lon,lat,'.r','MarkerSize',20)
%    plot_google_map
%
% References:
%  http://www.mathworks.com/matlabcentral/fileexchange/24113
%  http://www.maptiler.org/google-maps-coordinates-tile-bounds-
projection/
%  http://developers.google.com/maps/documentation/staticmaps/
%
% Acknowledgements:
%  Val Schmidt for his submission of get_google_map.m
%
% Author:
%  Zohar Bar-Yehuda
%
% Version 1.8 - 25/04/2016 - By Hannes Diethelm
%       - Add resize parameter to resize image using imresize()
%       - Fix scale parameter
% Version 1.7 - 14/04/2016
%       - Add custom style support
% Version 1.6 - 12/11/2015
%       - Use system temp folder for writing image files (with fallback
to current dir if missing write permissions)
% Version 1.5 - 20/11/2014
%       - Support for MATLAB R2014b
```

```
%       - several fixes for complex layouts: several maps in one
figure,
%         map inside a panel, specifying axis handle as input (thanks
to Luke Plausin)
% Version 1.4 - 25/03/2014
%       - Added the language parameter for showing labels in a local
language
%       - Display the URL on error to allow easier debugging of API
errors
% Version 1.3 - 06/10/2013
%       - Improved functionality of AutoAxis, which now handles any
shape of map axes.
%         Now also updates the extent of the map if the figure is
resized.
%       - Added the showLabels parameter which allows hiding the
textual labels on the map.
% Version 1.2 - 16/06/2012
%       - Support use of the "scale=2" parameter by default for finer
rendering (set scale=1 if too slow).
%       - Auto-adjust axis extent so the map isn't stretched.
%       - Set and use an API key which enables a much higher usage
volume per day.
% Version 1.1 - 25/08/2011

persistent apiKey useTemp
if isnumeric(apiKey)
    % first run, check if API key file exists
    if exist('api_key.mat','file')
        load api_key
    else
        apiKey = '';
    end
end

if isempty(useTemp)
    % first run, check we we have wrtie access to the temp folder
    try
        tempfilename = tempname;
        fid = fopen(tempfilename, 'w');
        if fid > 0
            fclose(fid);
            useTemp = true;
            delete(tempfilename);
        else
            % Don't have write access to temp folder or it doesn't
exist, fallback to current dir
            useTemp = false;
        end
    catch
        % in case tempname fails for some reason
        useTemp = false;
    end
end

hold on
```

```
% Default parametrs
axHandle = gca;
height = 640;
width = 640;
scale = 2;
resize = 1;
maptype = 'roadmap';
alphaData = 1;
autoRefresh = 1;
figureResizeUpdate = 1;
autoAxis = 1;
showLabels = 1;
language = '';
markeridx = 1;
markerlist = {};
style = '';

% Handle input arguments
if nargin >= 2
    for idx = 1:2:length(varargin)
        switch lower(varargin{idx})
            case 'axis'
                axHandle = varargin{idx+1};
            case 'height'
                height = varargin{idx+1};
            case 'width'
                width = varargin{idx+1};
            case 'scale'
                scale = round(varargin{idx+1});
                if scale < 1 || scale > 2
                    error('Scale must be 1 or 2');
                end
            case 'resize'
                resize = varargin{idx+1};
            case 'maptype'
                maptype = varargin{idx+1};
            case 'alpha'
                alphaData = varargin{idx+1};
            case 'refresh'
                autoRefresh = varargin{idx+1};
            case 'showlabels'
                showLabels = varargin{idx+1};
            case 'figureresizeupdate'
                figureResizeUpdate = varargin{idx+1};
            case 'language'
                language = varargin{idx+1};
            case 'marker'
                markerlist{markeridx} = varargin{idx+1};
                markeridx = markeridx + 1;
            case 'autoaxis'
                autoAxis = varargin{idx+1};
            case 'apikey'
                apiKey = varargin{idx+1}; % set new key
                % save key to file
```

```
                funcFile = which('plot_google_map.m');
                pth = fileparts(funcFile);
                keyFile = fullfile(pth,'api_key.mat');
                save(keyFile,'apiKey')
            case 'style'
                style = varargin{idx+1};
            otherwise
                error(['Unrecognized variable: ' varargin{idx}])
        end
    end
end
if height > 640
    height = 640;
end
if width > 640
    width = 640;
end

% Store paramters in axis handle (for auto refresh callbacks)
ud = get(axHandle, 'UserData');
if isempty(ud)
    % explicitly set as struct to avoid warnings
    ud = struct;
end
ud.gmap_params = varargin;
set(axHandle, 'UserData', ud);

curAxis = axis(axHandle);
if max(abs(curAxis)) > 500
    return;
end

% Enforce Latitude constraints of EPSG:900913
if curAxis(3) < -85
    curAxis(3) = -85;
end
if curAxis(4) > 85
    curAxis(4) = 85;
end
% Enforce longitude constrains
if curAxis(1) < -180
    curAxis(1) = -180;
end
if curAxis(1) > 180
    curAxis(1) = 0;
end
if curAxis(2) > 180
    curAxis(2) = 180;
end
if curAxis(2) < -180
    curAxis(2) = 0;
end

if isequal(curAxis,[0 1 0 1]) % probably an empty figure
    % display world map
```

```
    curAxis = [-200 200 -85 85];
    axis(curAxis)
end


if autoAxis
    % adjust current axis limit to avoid strectched maps
    [xExtent,yExtent] = latLonToMeters(curAxis(3:4), curAxis(1:2) );
    xExtent = diff(xExtent); % just the size of the span
    yExtent = diff(yExtent);
    % get axes aspect ratio
    drawnow
    org_units = get(axHandle,'Units');
    set(axHandle,'Units','Pixels')
    ax_position = get(axHandle,'position');
    set(axHandle,'Units',org_units)
    aspect_ratio = ax_position(4) / ax_position(3);

    if xExtent*aspect_ratio > yExtent
        centerX = mean(curAxis(1:2));
        centerY = mean(curAxis(3:4));
        spanX = (curAxis(2)-curAxis(1))/2;
        spanY = (curAxis(4)-curAxis(3))/2;

        % enlarge the Y extent
        spanY = spanY*xExtent*aspect_ratio/yExtent; % new span
        if spanY > 85
            spanX = spanX * 85 / spanY;
            spanY = spanY * 85 / spanY;
        end
        curAxis(1) = centerX-spanX;
        curAxis(2) = centerX+spanX;
        curAxis(3) = centerY-spanY;
        curAxis(4) = centerY+spanY;
    elseif yExtent > xExtent*aspect_ratio

        centerX = mean(curAxis(1:2));
        centerY = mean(curAxis(3:4));
        spanX = (curAxis(2)-curAxis(1))/2;
        spanY = (curAxis(4)-curAxis(3))/2;
        % enlarge the X extent
        spanX = spanX*yExtent/(xExtent*aspect_ratio); % new span
        if spanX > 180
            spanY = spanY * 180 / spanX;
            spanX = spanX * 180 / spanX;
        end

        curAxis(1) = centerX-spanX;
        curAxis(2) = centerX+spanX;
        curAxis(3) = centerY-spanY;
        curAxis(4) = centerY+spanY;
    end
    % Enforce Latitude constraints of EPSG:900913
    if curAxis(3) < -85
```

```
        curAxis(3:4) = curAxis(3:4) + (-85 - curAxis(3));
    end
    if curAxis(4) > 85
        curAxis(3:4) = curAxis(3:4) + (85 - curAxis(4));
    end
    axis(axHandle, curAxis); % update axis as quickly as possible,
before downloading new image
    drawnow
end

% Delete previous map from plot (if exists)
if nargout <= 1 % only if in plotting mode
    curChildren = get(axHandle,'children');
    map_objs = findobj(curChildren,'tag','gmap');
    bd_callback = [];
    for idx = 1:length(map_objs)
        if ~isempty(get(map_objs(idx),'ButtonDownFcn'))
            % copy callback properties from current map
            bd_callback = get(map_objs(idx),'ButtonDownFcn');
        end
    end
    delete(map_objs)

end

% Calculate zoom level for current axis limits
[xExtent,yExtent] = latLonToMeters(curAxis(3:4), curAxis(1:2) );
minResX = diff(xExtent) / width;
minResY = diff(yExtent) / height;
minRes = max([minResX minResY]);
tileSize = 256;
initialResolution = 2 * pi * 6378137 / tileSize; % 156543.03392804062
for tileSize 256 pixels
zoomlevel = floor(log2(initialResolution/minRes));

% Enforce valid zoom levels
if zoomlevel < 0
    zoomlevel = 0;
end
if zoomlevel > 19
    zoomlevel = 19;
end

% Calculate center coordinate in WGS1984
lat = (curAxis(3)+curAxis(4))/2;
lon = (curAxis(1)+curAxis(2))/2;

% Construct query URL
preamble = 'http://maps.googleapis.com/maps/api/staticmap';
location = ['?center=' num2str(lat,10) ',' num2str(lon,10)];
zoomStr = ['&zoom=' num2str(zoomlevel)];
sizeStr = ['&scale=' num2str(scale) '&size=' num2str(width) 'x'
num2str(height)];
maptypeStr = ['&maptype=' maptype ];
```

```
if ~isempty(apiKey)
    keyStr = ['&key=' apiKey];
else
    keyStr = '';
end
markers = '&markers=';
for idx = 1:length(markerlist)
    if idx < length(markerlist)
        markers = [markers markerlist{idx} '%7C'];
    else
        markers = [markers markerlist{idx}];
    end
end

if showLabels == 0
    if ~isempty(style)
        style(end+1) = '|';
    end
    style = [style 'feature:all|element:labels|visibility:off'];
end

if ~isempty(language)
    languageStr = ['&language=' language];
else
    languageStr = '';
end

if ismember(maptype,{'satellite','hybrid'})
    filename = 'tmp.jpg';
    format = '&format=jpg';
    convertNeeded = 0;
else
    filename = 'tmp.png';
    format = '&format=png';
    convertNeeded = 1;
end
sensor = '&sensor=false';

if ~isempty(style)
    styleStr = ['&style=' style];
else
    styleStr = '';
end

url = [preamble location zoomStr sizeStr maptypeStr format markers
languageStr sensor keyStr styleStr];

% Get the image
if useTemp
    filepath = fullfile(tempdir, filename);
else
    filepath = filename;
end
```

```
try
    urlwrite(url,filepath);
catch % error downloading map
    warning(['Unable to download map form Google Servers.\n' ...
        'Matlab error was: %s\n\n' ...
        'Possible reasons: missing write permissions, no network
connection, quota exceeded, or some other error.\n' ...
        'Consider using an API key if quota problems persist.\n\n' ...
        'To debug, try pasting the following URL in your browser, which
may result in a more informative error:\n%s'], lasterr, url);
    varargout{1} = [];
    varargout{2} = [];
    varargout{3} = [];
    return
end
[M Mcolor] = imread(filepath);
M = cast(M,'double');
delete(filepath); % delete temp file
width = size(M,2);
height = size(M,1);

% We now want to convert the image from a colormap image with an uneven
% mesh grid, into an RGB truecolor image with a uniform grid.
% This would enable displaying it with IMAGE, instead of PCOLOR.
% Advantages are:
% 1) faster rendering
% 2) makes it possible to display together with other colormap
annotations (PCOLOR, SCATTER etc.)

% Convert image from colormap type to RGB truecolor (if PNG is used)
if convertNeeded
    imag = zeros(height,width,3);
    for idx = 1:3
        imag(:,:,idx) = reshape(Mcolor(M(:)+1+(idx-
1)*size(Mcolor,1)),height,width);
    end
else
    imag = M/255;
end
% Resize if needed
if resize ~= 1
    imag = imresize(imag, resize, 'bilinear');
end

% Calculate a meshgrid of pixel coordinates in EPSG:900913
width = size(imag,2);
height = size(imag,1);
centerPixelY = round(height/2);
centerPixelX = round(width/2);
[centerX,centerY] = latLonToMeters(lat, lon ); % center coordinates in
EPSG:900913
curResolution = initialResolution / 2^zoomlevel / scale / resize; %
meters/pixel (EPSG:900913)
xVec = centerX + ((1:width)-centerPixelX) * curResolution; % x vector
```

```
yVec = centerY + ((height:-1:1)-centerPixelY) * curResolution; % y
vector
[xMesh,yMesh] = meshgrid(xVec,yVec); % construct meshgrid

% convert meshgrid to WGS1984
[lonMesh,latMesh] = metersToLatLon(xMesh,yMesh);

% Next, project the data into a uniform WGS1984 grid
uniHeight = round(height*resize);
uniWidth = round(width*resize);
latVect = linspace(latMesh(1,1),latMesh(end,1),uniHeight);
lonVect = linspace(lonMesh(1,1),lonMesh(1,end),uniWidth);
[uniLonMesh,uniLatMesh] = meshgrid(lonVect,latVect);
uniImag = zeros(uniHeight,uniWidth,3);

% old version (projection using INTERP2)
% for idx = 1:3
%      % 'nearest' method is the fastest. difference from other methods
is neglible
%          uniImag(:,:,idx) =
interp2(lonMesh,latMesh,imag(:,:,idx),uniLonMesh,uniLatMesh,'nearest');
% end
uniImag =  myTurboInterp2(lonMesh,latMesh,imag,uniLonMesh,uniLatMesh);

if nargout <= 1 % plot map
    % display image
    hold(axHandle, 'on');
    cax = caxis;
    h = image(lonVect,latVect,uniImag, 'Parent', axHandle);
    caxis(cax); % Preserve caxis that is sometimes changed by the call
to image()
    set(axHandle,'YDir','Normal')
    set(h,'tag','gmap')
    set(h,'AlphaData',alphaData)

    % add a dummy image to allow pan/zoom out to x2 of the image extent
    h_tmp = image(lonVect([1 end]),latVect([1
end]),zeros(2),'Visible','off', 'Parent', axHandle);
    set(h_tmp,'tag','gmap')

    % older version (display without conversion to uniform grid)
    % h =pcolor(lonMesh,latMesh,(M));
    % colormap(Mcolor)
    % caxis([0 255])
    % warning off % to avoid strange rendering warnings
    % shading flat

    uistack(h,'bottom') % move map to bottom (so it doesn't hide
previously drawn annotations)
    axis(axHandle, curAxis) % restore original zoom
    if nargout == 1
        varargout{1} = h;
    end
```

```
    % if auto-refresh mode - override zoom callback to allow autumatic
    % refresh of map upon zoom actions.
    figHandle = axHandle;
    while ~strcmpi(get(figHandle, 'Type'), 'figure')
        % Recursively search for parent figure in case axes are in a
panel
        figHandle = get(figHandle, 'Parent');
    end

    zoomHandle = zoom(axHandle);
    panHandle = pan(figHandle); % This isn't ideal, doesn't work for
contained axis
    if autoRefresh
        set(zoomHandle,'ActionPostCallback',@update_google_map);
        set(panHandle, 'ActionPostCallback', @update_google_map);
    else % disable zoom override
        set(zoomHandle,'ActionPostCallback',[]);
        set(panHandle, 'ActionPostCallback',[]);
    end

    % set callback for figure resize function, to update extents if
figure
    % is streched.
    if figureResizeUpdate &&isempty(get(figHandle, 'ResizeFcn'))
        % set only if not already set by someone else
        set(figHandle, 'ResizeFcn', @update_google_map_fig);
    end

    % set callback properties
    set(h,'ButtonDownFcn',bd_callback);
else % don't plot, only return map
    varargout{1} = lonVect;
    varargout{2} = latVect;
    varargout{3} = uniImag;
end

% Coordinate transformation functions

function [lon,lat] = metersToLatLon(x,y)
% Converts XY point from Spherical Mercator EPSG:900913 to lat/lon in
WGS84 Datum
originShift = 2 * pi * 6378137 / 2.0; % 20037508.342789244
lon = (x ./ originShift) * 180;
lat = (y ./ originShift) * 180;
lat = 180 / pi * (2 * atan( exp( lat * pi / 180)) - pi / 2);

function [x,y] = latLonToMeters(lat, lon )
% Converts given lat/lon in WGS84 Datum to XY in Spherical Mercator
EPSG:900913"
originShift = 2 * pi * 6378137 / 2.0; % 20037508.342789244
x = lon * originShift / 180;
y = log(tan((90 + lat) * pi / 360 )) / (pi / 180);
y = y * originShift / 180;
```

```
function ZI = myTurboInterp2(X,Y,Z,XI,YI)
% An extremely fast nearest neighbour 2D interpolation, assuming both
input
% and output grids consist only of squares, meaning:
% - uniform X for each column
% - uniform Y for each row
XI = XI(1,:);
X = X(1,:);
YI = YI(:,1);
Y = Y(:,1);

xiPos = nan*ones(size(XI));
xLen = length(X);
yiPos = nan*ones(size(YI));
yLen = length(Y);
% find x conversion
xPos = 1;
for idx = 1:length(xiPos)
    if XI(idx) >= X(1) && XI(idx) <= X(end)
        while xPos < xLen && X(xPos+1)<XI(idx)
            xPos = xPos + 1;
        end
        diffs = abs(X(xPos:xPos+1)-XI(idx));
        if diffs(1) < diffs(2)
            xiPos(idx) = xPos;
        else
            xiPos(idx) = xPos + 1;
        end
    end
end
% find y conversion
yPos = 1;
for idx = 1:length(yiPos)
    if YI(idx) <= Y(1) && YI(idx) >= Y(end)
        while yPos < yLen && Y(yPos+1)>YI(idx)
            yPos = yPos + 1;
        end
        diffs = abs(Y(yPos:yPos+1)-YI(idx));
        if diffs(1) < diffs(2)
            yiPos(idx) = yPos;
        else
            yiPos(idx) = yPos + 1;
        end
    end
end
ZI = Z(yiPos,xiPos,:);


function update_google_map(obj,evd)
% callback function for auto-refresh
drawnow;
try
    axHandle = evd.Axes;
catch ex
```

```
    % Event doesn't contain the correct axes. Panic!
    axHandle = gca;
end
ud = get(axHandle, 'UserData');
if isfield(ud, 'gmap_params')
    params = ud.gmap_params;
    plot_google_map(params{:});
end


function update_google_map_fig(obj,evd)
% callback function for auto-refresh
drawnow;
axes_objs = findobj(get(gcf,'children'),'type','axes');
for idx = 1:length(axes_objs)
    if ~isempty(findobj(get(axes_objs(idx),'children'),'tag','gmap'));
        ud = get(axes_objs(idx), 'UserData');
        if isfield(ud, 'gmap_params')
            params = ud.gmap_params;
        else
            params = {};
        end

        % Add axes to inputs if needed
        if ~sum(strcmpi(params, 'Axis'))
            params = [params, {'Axis', axes_objs(idx)}];
        end
        plot_google_map(params{:});
    end
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. PYTHON SCRIPT

The following Python script was used to convert the .bin data flash log files generated by the PIXHAWK PX4 to .m files that could be imported into MATLAB for analysis. It is available as part of the ArduPilot software suite. While it was not written by the author of this thesis, no one author can be named as it is an ongoing collaborative effort. It can be found at the following web address:

https://github.com/ArduPilot/pymavlink

```
#!/usr/bin/env python

'''
convert a MAVLink tlog file to a MATLab mfile
'''
from __future__ import print_function
from builtins import range

import os
import re
from pymavlink import mavutil

def process_tlog(filename):
    '''convert a tlog to a .m file'''

    print("Processing %s" % filename)

    mlog = mavutil.mavlink_connection(filename, dialect=args.dialect,
zero_time_base=True)

    # first walk the entire file, grabbing all messages into a hash of lists,
    #and the first message of each type into a hash
    msg_types = {}
    msg_lists = {}

    types = args.types
    if types is not None:
        types = types.split(',')

    # note that Octave doesn't like any extra '.', '*', '-', characters in the filename
    (head, tail) = os.path.split(filename)
```

```
basename = '.'.join(tail.split('.')[:-1])
mfilename = re.sub('[\.\-\+\*]','_', basename) + '.m'
# Octave also doesn't like files that don't start with a letter
if (re.match('^[a-zA-z]', mfilename) == None):
    mfilename = 'm_' + mfilename

if head is not None:
    mfilename = os.path.join(head, mfilename)
print("Creating %s" % mfilename)

f = open(mfilename, "w")

type_counters = {}

while True:
    m = mlog.recv_match(condition=args.condition)
    if m is None:
        break

    if types is not None and m.get_type() not in types:
        continue
    if m.get_type() == 'BAD_DATA':
        continue

    fieldnames = m._fieldnames
    mtype = m.get_type()
    if mtype in ['FMT', 'PARM']:
        continue

    if mtype not in type_counters:
        type_counters[mtype] = 0
        f.write("%s.columns = {'timestamp"' % mtype)
        for field in fieldnames:
            val = getattr(m, field)
            if not isinstance(val, str):
                if type(val) is not list:
                    f.write(",'%s"' % field)
                else:
                    for i in range(0, len(val)):
                        f.write(",'%s%d"' % (field, i + 1))
        f.write("};\n")

    type_counters[mtype] += 1
    f.write("%s.data(%u,:) = [%f" % (mtype, type_counters[mtype], m._timestamp))
    for field in m._fieldnames:
```

```
            val = getattr(m, field)
            if not isinstance(val, str):
                if type(val) is not list:
                    f.write(",%.20g" % val)
                else:
                    for i in range(0, len(val)):
                        f.write(",%.20g" % val[i])
        f.write("];\n")
    f.close()

from argparse import ArgumentParser
parser = ArgumentParser(description=__doc__)

parser.add_argument("--condition", default=None, help="select packets by condition")
parser.add_argument("-o", "--output", default=None, help="output filename")
parser.add_argument("--types", default=None, help="types of messages (comma
separated)")
parser.add_argument("--dialect", default="ardupilotmega", help="MAVLink dialect")
parser.add_argument("logs", metavar="LOG", nargs="+")
args = parser.parse_args()


for filename in args.logs:
    process_tlog(filename)
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Arcturus UAV. "T-20." Accessed October 10, 2016. http://arcturusuav.com/product/t-20.

Beall, Ryan, and Chaz Henderson. 2016. "Rapid Prototyping and Quick System Identification." *DIY Drones*.

Benney, Richard, Justin Barber, Joseph McGrath, Jaclyn McHugh, Greg Noetscher, and Steve Tavan. 2005. "The Joint Precision Airdrop System Advanced Concept Technology Demonstration." Paper presented at the 18th AIAA Aerodynamic Decelerator Systems Technology Conference and Seminar, Munich, Germany, May 24–26.

———. 2005. "The New Military Applications of Precision Airdrop Systems." Paper presented at AIAA Infotech@Aerospace Conference, Arlington, Virginia, September 26–29.

Benney, Richard, Joseph McGrath, Jaclyn McHugh, Andrew Meloni, Greg Noetscher, Steve Tavan and Sanjay Patel. 2007. "DOD JPADS Programs Overview and NATO Activities." Paper presented at the 19th AIAA Aerodynamic Decelerator Systems Technology Conference and Seminar, Williamsburg, VA, May 21–24.

Brown, Glen, and Richard Benney. 2005. "Precision Aerial Delivery Systems in a Tactical Environment." Paper presented at the 18th AIAA Aerodynamic Decelerator Systems Technology Conference and Seminar, Munich, Germany, May 24–26.

Center for Interdisciplinary Remotely-Piloted Aircraft Studies (CIRPAS). "McMillan Facilities." Accessed October 29, 2016a. http://www.cirpas.org/facilities.html/

_____. "Airspace, Special Use Airspace, and TFRs." Accessed October 29, 2016b. https://www.faasafety.gov/gslac/ALC/course_content.aspx?cID=42&sID=242&preview=true/

Defense Industry Daily. 2014. "JPADS: Making Precision Airdrop a Reality." Accessed on March 21, 2016. http://www.defenseindustrydaily.com/jpads-makingprecision-airdrop-a-reality-0678/.

DJI. "Inspire 1." Accessed October 28, 2016. http://www.dji.com/inspire-1/aircraft/

Flite Test. "FT Pun Jet Build." Accessed August 15, 2015. http://www.flitetest.com/articles/ft-pun-jet-build/

Flite Test. "FT Sparrow Build." Accessed March 1, 2016. http://www.flitetest.com/articles/ft-sparrow-build/

Hall, Andrew. “Conceptual and Preliminary Design of a Low-Cost Precision Aerial Delivery System.” Master’s thesis, Naval Postgraduate School, 2016.

Marine Corps Warfighting Laboratory (MCWL). 2013. “Performance Work Statement (PWS) for Prototype Tactical Air Delivery (TACAD) Design Characterization.” Accessed on October 28, 2016.

Marine Corps Warfighting Laboratory (MCWL). 2015. “Tactical Air Delivery (TACAD) Cost Benefit Analysis.” Accessed on October 28, 2016.

O’Brian, Matthew. “Wind Estimation for Aerial Payload Delivery Systems Using GPS and IMU.” Master’s thesis, Naval Postgraduate School, 2016.

Office of Naval Research (ONR). 2012. “Autonomous Aerial Cargo/Utility System (AACUS) Innovative Naval Prototype (INP) Concept of Operations (CONOPS).” Accessed September 24, 2016. http://www.onr.navy.mil/~/media/Files/Funding-Announcements/BAA/2012/12-004-CONOPS.

PIXHAWK. “Pixhawk Autopilot.” Accessed November 1, 2016. https://pixhawk.org/modules/pixhawk

Tavan, Steve. 2006. “Status and Context of High Altitude Precision Aerial Delivery Systems.” Paper presented to the AIAA Guidance, Navigation, and Control Conference and Exhibit, Keystone, CO, August 21–24.

United States Marine Corps (USMC). 2013. “Marine Corps Installations and Logistics Roadmap.” Accessed May 21, 2016. https://marinecorpsconceptsandprograms.com/sites/default/files/files/Marine%20Corps%20Installations%20and%20Logistics%20Roadmap.pdf

Yakimenko, Oleg A. 2015. “PADS and Measures of Their Effectiveness.” In *Precision Aerial Delivery Systems: Modeling, Dynamics, and Control*, edited by Oleg A. Yakimenko. Reston, VA: American Institute of Aeronautics and Astronautics, Inc.

Yakimenko, Oleg, Eugene Bourakov, Charles Hewgley, Nathan Slegers, Red Jensen, Andrew Robinson, Josh Malone, and Phil Heidt. 2011. “Autonomous Aerial Payload Delivery System ‘Blizzard.’” Paper presented at the 21st AIAA Aerodynamic Decelerator Systems Technology Conference and Seminar, Dublin, Ireland, May 23–26.

Yakimenko, Oleg. 2016. “Autonomous Parachute-Based Precision Delivery Systems.” *Encyclopedia of Aerospace Engineering*. Hoboken, NJ: John Wiley & Sons.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California